Manuel d'utilisation

Script

DTE/037F • V1.2 • 02/2022

SOMMAIRE

1. Présentation

Généralités

Les ULI e@sy et REDY intègrent un générateur de Scripts qui se développent dans un langage évolué proche du Pascal.

Les Scripts permettent de créer des process particulier, de gérer la réception ou l'envoi de SMS et d'e-mails, de créer un protocole de communication spécifique ou de dynamiser l'affichage sur une imagerie.

Prérequis

Dans le cas d'un automate e@sy, il est nécessaire de posséder une version logicielle permettant l'utilisation de ressources de type Script.

e@sy-pro et e@sy-IO : Version + ou Version ++

Domaines d'application

Les Ressources

Script

La ressource « Script » permet de réaliser des **process** spécifiques ainsi que des **calculs complexes**.

Script Driver

La ressource « Script Driver » permet de développer son propre **protocole de communication** en offrant la possibilité de gérer la réception et l'émission de données sur une liaison série (RS232/RS485) ou IP.

Script SMS

La ressource « Script SMS » permet à un une personne d'**interagir avec une ULI par SMS**[1] : interrogation d'états, acquittement d'alarmes, ordre de commande, etc.

Script email

La ressource « Script email » permet de gérer la **réception d'e-mails** depuis un serveur de messagerie POP.

Les Synoptiques

Script Synoptique

Le « Script Synoptique »

[1] Nécessite un e@sy intégrant un modem GSM (PLUG GSM), une Extension GSM Cube ou un REDY modem embarqué.

Avertissement

Une mauvaise utilisation des Scripts peut altérer le bon fonctionnement des ULI e@sy ou REDY :

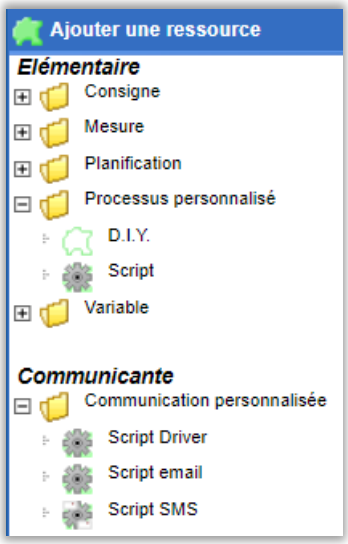
- ralentissement de la navigation web et des automatismes
- saturation de la mémoire vive (RAM)
- blocage ou redémarrage (ex : un script bouclant indéfiniment sur lui-même)

Afin d'acquérir une parfaite connaissance et maîtrise des scripts, il est conseillé de bénéficier d'une formation « WIT Expert ».

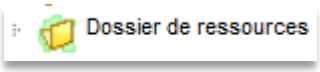
Pour connaître le programme et les dates de nos formations, vous pouvez nous écrire sur wit@wit.fr.

2. Création de la ressource

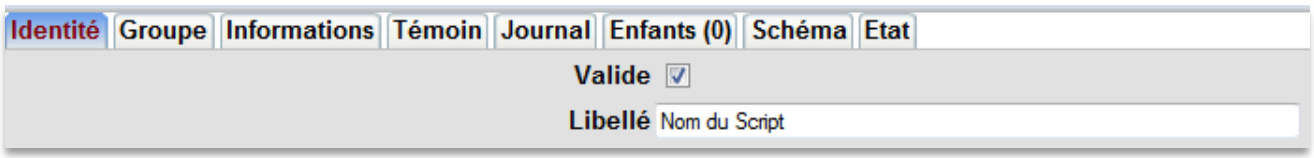
Etape 1 Insérer la ressource Script correspondante depuis le menu Paramétrage ► Ressource ► Ajouter une ressource ► Processus personnalisé ou Communication personnalisée.



Dans un souci de clarté dans l'arborescence des ressources, il est conseillé de créer les scripts dans un dossier de ressource dont le nom pourra être en rapport avec les scripts qu'il contient :



Etape 2 Nommer la ressource Script :

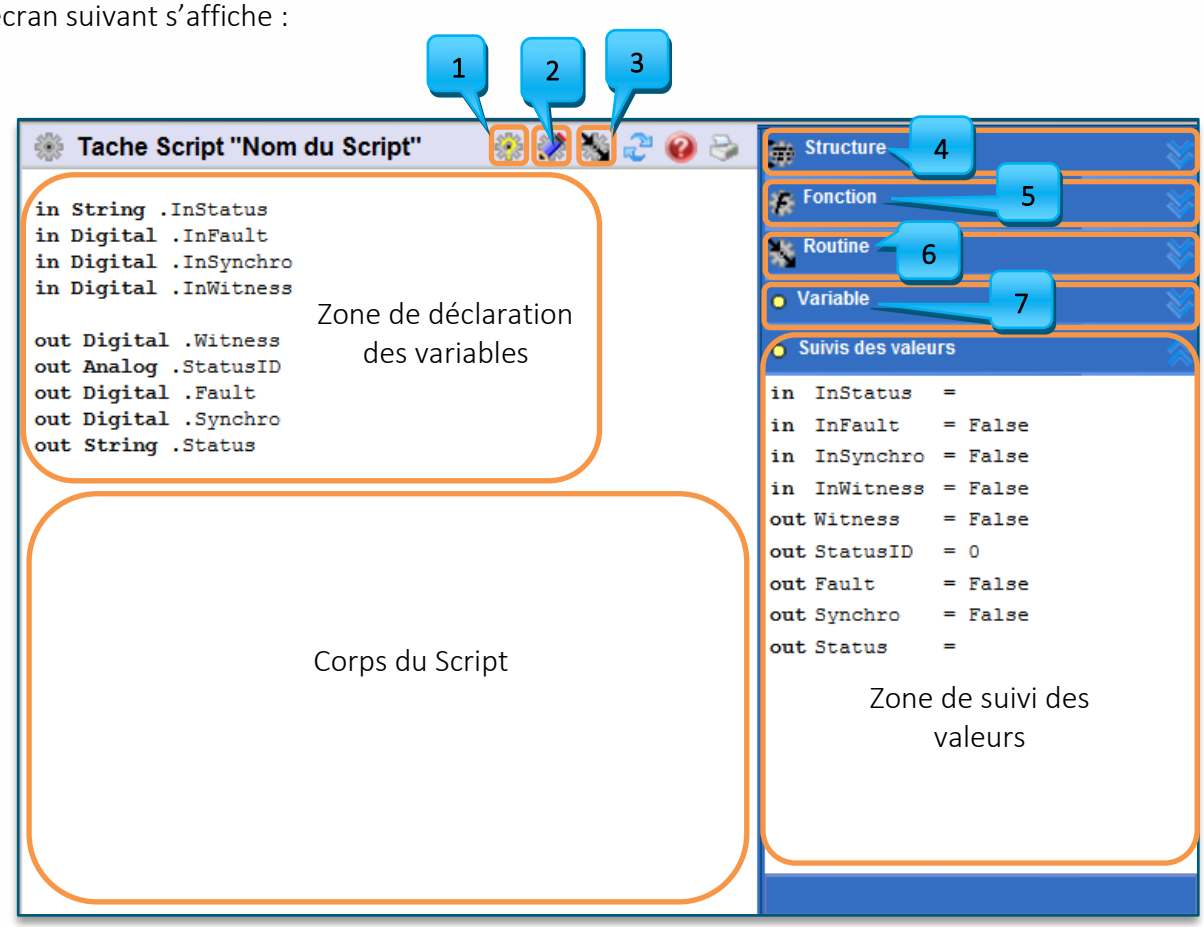


3. Paramétrage

Edition

Pour passer en mode édition du script cliquer sur la roue crantée, bouton **Script...** , dans les paramètres de la ressource :

L'écran suivant s'affiche :



1 Information sur le script
2 Mode d'édition
3 Routine
4 Structures disponibles
5 Fonctions disponibles
6 Routines disponibles
7 Variables internes disponibles

Définitions

La déclaration des Variables

- Les variables peuvent être de plusieurs types :
  - Booléenne (Digital)
  - Entière ou réelle (Analog)
  - Chaîne (String) limitée à 255 caractères
  - Fichier (Blob)

- Elles peuvent être classées en plusieurs catégories :
  - **Variables d'entrée** (Ex : .InStatus) Elles peuvent être utilisées en entrée de script pour du passage de paramètres d'entrée, utiles au fonctionnement du script : passage de la valeur d'une entrée, pour le déclenchement d'une action.

  Elles sont notées *in String, in Digital, in Analog* et elles commencent par un « **.** ».

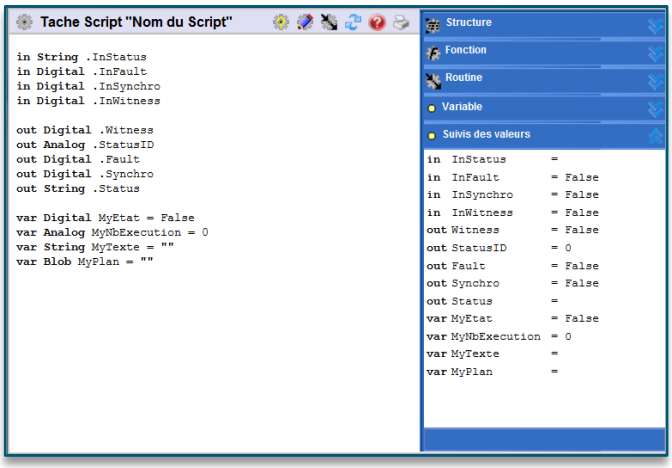
  - **Variables de sortie** (Ex : .StatusID) Elles peuvent être utilisées pour des états liés au fonctionnement de la ressource :

.StatusID fournit l'état du flag de la ressource (Dévalidée, En alarme non acquittée,….)
.WitNess à mettre à 1 pour générer un évènement sur cette ressource.
.Status pour inscrire l'état de la Ressource dans le cas d'évènements.
.Fault pour signaliser que la ressource est en défaut.

Elles sont notées *out String, out Digital, out Analog* et elles commencent par un « **.** ».

- **Variables du script**. Elles sont utilisées dans le script et restent locales à ce script.

Elles sont notées *Var String, Var Digital, Var Analog et Var Blob* et elles commencent par « **My** ». Le « . » n'est pas utilisé :

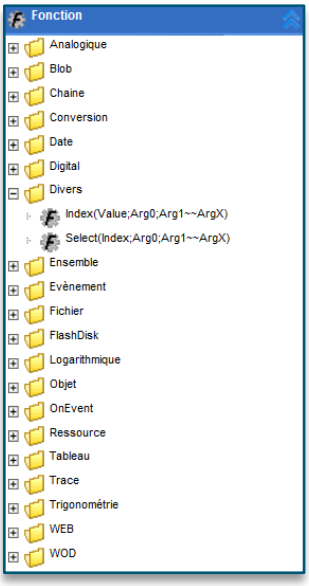


La zone de déclaration des variables apparaît en tête du corps du script. Les mêmes variables se retrouvent dans la zone de 'Suivis des valeurs', ce qui permet de suivre leur évolution au cours du déroulement du script.

Interface

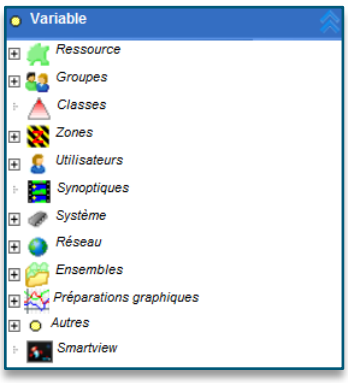
- Structure : cet onglet contient toutes les structures de programmation que l'on peut insérer par un glissé/déposé.

- Fonction : cet onglet contient toutes les fonctions disponible pour le traitement d'une valeur, les calculs etc…



- Routine : Par défaut, cet onglet est vide, c'est ici que se trouverons les routines créées, ces routines sont des petits programmes, elles viennent compléter les actions qui ne sont pas réalisées par les fonctions déjà existantes.

- Variable : Explorateur de ressources de l'ULI :



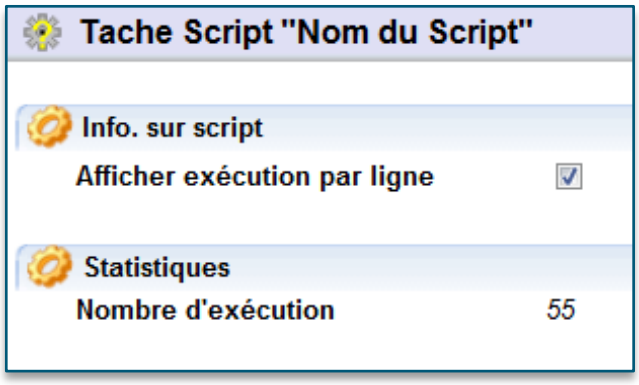
Nota : Pour améliorer la lisibilité des scripts, il est possible de rajouter des lignes de commentaires. Une ligne de commentaire débute par un double slash « // » et est repérée en bleu dans le script :

```
if .InStatus then
  // Affectation de zone
  MyIndex = 1
  while MyIndex<16 do
    @":easy.RESS.ExtenBUS.EXT001.DI"
    MyIndex = MyIndex+1
  end
  .InStatus = 0
end
```

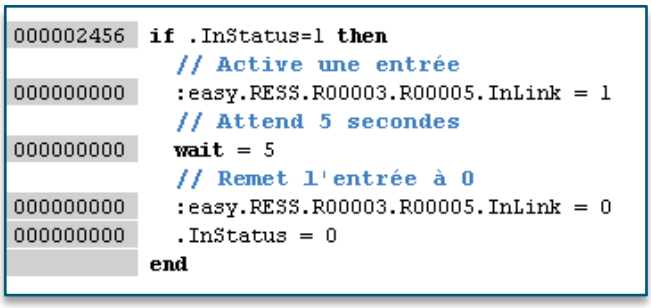
Mise au point du script

Pour afficher les informations sur le déroulement du script, il est nécessaire de cliquer sur le bouton Info. sur script :

La fenêtre suivante s'affiche :

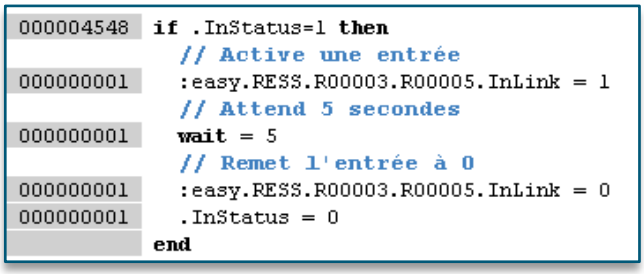


Les statistiques donnent le nombre d'exécution du Script.
La coche **Afficher exécution par ligne** permet de comptabiliser le nombre de passage sur chaque ligne de script :



Le script a eu la main 2456 fois, sur les autres lignes, le script n'a pas été exécuté sur les autres lignes.

Si **.InStatus** est à 1, on obtient alors :



Le Script a été exécuté une seule fois.

4. Avant de commencer

Il faut savoir :

- Qu'est-ce qu'on veut faire ?
- Définir les variables d'entrées.
- Définir les variables de sorties.
- Quand exécuter ce script ?
- Utiliser des variables locales.
- Repérer grâce à l'explorateur, le nom des propriétés à utiliser.
- Nommer intelligemment les variables.

5. La syntaxe

Les constantes

Les constantes dans les scripts sont précédées du caractère #.
Listes des constantes:

- 'True';'False';'Pi';'e';'Paste';'Futur';'NUL';'SOH';'STX';'ETX';'EOT';'ENQ';'ACK';
- 'BEL';'BS';'HT';'LF';'VT';'FF';'CR';'SO';'SI';'DLE';
- 'DC1';'DC2';'DC3';'DC4';'NAK';'SYN';'ETB';'CAN';'EM';'SUB';
- 'ESC';'FS';'GS';'RS';'US';'SPACE';'DEL';'TAB';
- 'MaskDate';'MaskTime';'MaskDateTime';'MaskPhone';
- 'MaskZipCode';'MaskCB';'MaskSecu';
- 'Boolean';'Small';'Short';'Long';'Single';'Double';'String';

Exemple : *:easy.ress.r0001.InLink = #True*

Les opérateurs et comparateurs

Les opérateurs logiques sont les suivants :

- & : AND logique
- | : Or logique

Exemple : if OnDrawBack|OnRefreshActor then

Les comparaisons s'effectueront avec les fonctions suivantes :

- = : Egale *(Permet de comparer dans une structure if, Repeat ou While , ou d'affecter directement une valeur a un chemin)*
- <> : Différant
- <= : Inférieur ou égale
- >= : Supérieur ou égale

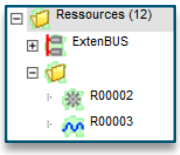
Exemple : if MyAnalog<=5 then

Pour concaténer 2 chaînes il est nécessaire d'utiliser les caractères && :

Exemple : MyStr="Heure = "&&Time(Clock)

Chemin ou Path

Les variables dans un script sont toujours repérées par leur chemin, sauf les variables dites locales (MyVar). Le chemin est composé des **LABEL** de tout dossier ou ressource.



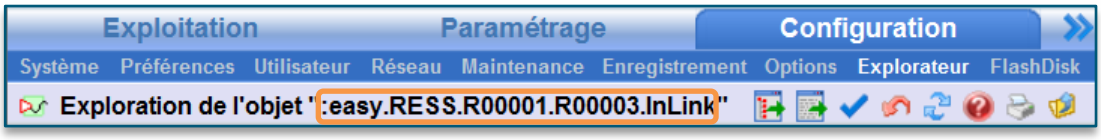
Chemin relatif: Son chemin est relatif par rapport au script.

Le script étant dans le même dossier que la ressource, son chemin sera donc : ..R00002.InLink = 54

Chemin absolu: Son chemin est entièrement décrit par rapport à la racine.

*La ressource se trouvant dans un dossier qui est dans **ressources**, le chemin sera donc : :easy.RESS.R00001.R00003.InLink = 54*

Pour trouver un chemin absolu complet, ou simplement le LABEL d'un dossier, il est nécessaire de consulter la page **Configuration ► Explorateur ► easy ► RESS ►** (Cliquer ensuite sur le dossier concerné puis les ressources enfants si présentes). Enfin, copier le lien absolu complet :



Le script étant enfant de la ressource dossier R00001, il est préférable d'écrire en relatif, dans le cas où l'on veut dupliquer le dossier. Les liens relatif seront toujours les mêmes dans le dossier dupliqué, le script dupliqué fonctionnera donc bien avec sa ressource associée, ce n'est pas le cas d'une adresse absolu.

Variable dynamique

Il arrive, dans certains cas, que les chemins d'accès aux objets soient calculés dynamiquement.
Par exemple, si on souhaite affecter la zone à toutes les ressources d'un module 15.0.0.0, il est possible d'effectuer la programmation suivante :

```
if .InStatus then
  :easy.RESS.ExtenBUS.EXT001.DI1.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI2.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI4.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI5.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI6.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI7.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI8.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI9.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI10.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI11.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI12.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI13.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI14.Zone = ":System.Attribute.Zone.Z_1"
  :easy.RESS.ExtenBUS.EXT001.DI15.Zone = ":System.Attribute.Zone.Z_1"
  .InStatus = 0
end
```

Une solution plus approprie consisterait à le programmer ainsi :

```
if .InStatus then
  MyIndex = 1
  while MyIndex<16 do
    @":easy.RESS.ExtenBUS.EXT001.DI"&&MyIndex&&".Zone" = ":System.Attribute.Zone.Z_1"
    MyIndex = MyIndex+1
  end
  .InStatus = 0
end
```

Le @ permet d'affecter une valeur dont le chemin de la variable est reconstitué :

@":easy.RESS.ExtenBUS.EXT001.DI"&&MyIndex&&".Zone" = ":System.Attribute.Zone.Z_1"

Pour récupérer la valeur d'une variable dont le chemin est reconstitué on utilise la fonction **Value .**

MyZone = Value(":easy.RESS.ExtenBUS.EXT001.DI"&&MyIndex&&".Zone")

Variables de Paramètres

Utilisées dans les routines. Elles s'appellent les « Arguments ».
Ce sont des paramètres passés à un script qui est exécuté une seule fois sur appel d'un script.
Elles sont notées *Arg String, Arg Digital, Arg Analog* et elles commencent par « **The** ». Le « . » n'est pas utilisé.
Une routine est appelée par un script pour effectuer un traitement spécifique.
Le script appelant passe généralement des arguments à la routine :

```
if .InStatus<>0 then
  // Modifie les Hint des Acteurs de signalisation
  Modif_Ress(":easy.SYN.SYN_20.SYN_22")
```

Le nom de la routine appelée est **Modif_Ress**.
L'argument est ici un chemin.

Dans la routine, nous retrouvons alors :

```
Routine Script "Modif_Ress(TheRacine)"

arg String TheRacine = ""
```

Le chemin est ici passé à l'argument **TheRacine**.

6. Les structures

Conditionelles

Il existe uniquement 4 types de structures conditionnelles

Instruction	Signification	Exemples

If x then ….end	Cette instruction permet d'exécuter les lignes de commandes entre le *then* et le *end* si la condition x est vrai. La condition x est booléenne.	If .InStatus=1 then
		// Lignes de commandes exécutées
		end

If x then… else…end	Cette instruction permet d'exécuter les lignes de commandes entre le *then* et le *else* si la condition x est vrai ou les lignes de commandes entre le *else* et le *end* si la condition *x* est fausse. La condition *x* est booléenne.	If .InStatus=1 then
		// Lignes de commandes exécutées si
		// .InStatus=1
		Else
		// Lignes de commandes exécutées si
		// .InStatus<>1
		end

Repeat
…
Until x
	Cette instruction permet d'exécuter en boucle les lignes entre le *Repeat* et le *Until* jusqu'à ce la condition *x* soit vraie. La condition *x* est booléenne.	Repeat
		// Lignes de commandes exécutées jusqu'à ce que la condition soit vraie
		Until :easy.RESS.R00006.Output=1

While x do
….
end
	Cette instruction permet d'exécuter en boucle les lignes entre le *Do* et le *End* tant que la condition *x* est vraie. La condition *x* est booléenne.	While :easy.RESS.R00006.Output=0 do
		// Lignes de commandes exécutées tant
		// que la condition est vraie

Spécifique

Le langage autorise également 3 autres structures non conditionnelles, qui permettent d'attendre, rebooter ou quitter le script.

Instruction	Signification
Break	Cette instruction permet de sortir du script sans exécuter les lignes qui suivent.
Wait = x	Cette instruction permet d'attendre x secondes avant d'effectuer les lignes suivantes du script. Le script rend la main au système, est rappelé au bout du temps x, et poursuit l'exécution.
Raz	Cette instruction permet de reboucler au début du script.

7. La liste des fonctions commune e@sy/REDY/ e@sy-Pilot

Affectation de valeurs

Pour affecter une valeur a une entrée, sortie ou variable, analogique ou digital, la valeur est directement inscrite après le signe égal.

MyAnalogue = 62
MyDIgit = 0

Pour affecter une valeur à une entrée, sortie ou variable texte ou blob, celle-ci est insérée entre double guillemet.

MyString = "Chaine de Caractère limiter à 255 caractères"
MyPlan = "Texte à entrer dans le blob

Les fonctions de traitement analogique

Fonction	Signification	Type du Résultat	Exemple
Min(Value1;Value2~~ValueX)	Retourne la valeur Minimale des valeurs saisies en Arguments	Réel	Min(:easy.RESS.R00003.R00006.Output;18)
Max(Value1;Value2~~ValueX)	Retourne la valeur Maximale des valeurs saisies en Arguments	Réel	
Odd(Value)	Retourne 'True' si l'argument est impair et 'False' dans le cas contraire	Booléen	Odd(9) = 1 Odd(8) = 0
Exp(Value[;Exponent])	Fonction Exposant	Réel	Exp(:easy.RESS.R00003.R00005.Output;2)
SquareRoot(Value)	Fonction Racine Carrée	Réel	
Absolute(Value)	Retourne la Valeur Absolue de l'Argument	Réel	Absolute(-6,4) = 6,4
Round(Value)	Retourne l'arrondi de l'Argument	Entier	Round(2,6) = 3 Round (-3,4) = -3
RoundHalfUp(Value[;Decimal])	Retourne la value arrondi avec décimal	Entier	RoundHalfUp(10.249;1)=10.2 RoundHalfUp(10.251;1) =10,3

Les fonctions de traitement des Blobs

Fonction	Signification	Résultat
BLOB("Object")	Sélectionne BLOB de travail	Result=True=Ok
BLOBLoad("ObjectSrc")	Load	Result=True=Ok
BLOBSave("ObjectDest")	Save	Result=True=Ok
BLOBClear	Efface le contenu du BLOB de travail	Result=True=Ok
BLOBCompact	Optimise le contenu du BLOB de travail pour ne pas tenir de place excessive en mémoire	Result=True=Ok
BLOBSize([NewSize])	Redimensionne la taille du BLOB de travail	Result=Taille du BLOB de travaille, (-1)=pas de BLOB
BLOBSwap("ObjectSwap")	Echange BLOB de travail avec "ObjectSwap",	Result=True=Ok
BLOBWrite(StrValue[;Position[;Len]]) Len=(Size-StartPosition)	Ajoute le text au BLOB de travail	Result=True=Ok
BLOBWriteFromBlob("BlobSrc"[;Position[;Len]])	Ajoute le contenu de BlobSrc au BLOB de travail	Result=True=Ok
BLOBInsert(StrValue[;Position[;Len]])	Insert StrValue dans BLOB de travail	Result=True=Ok
BLOBInsertFromBlob("BlobSrc"[;Position[;Len]])	Insert le contenu de BlobSrc dans BLOB de travail	Result=True=Ok
BLOBRead([Position[;Len]])		Result=Texte
BLOBReadToBlob("BlobDest"[;Position[;Len]])	Lecture du BLOB de travail dans BlobDest	Result=True=Ok
BLOBReadField(Index[;Sep]) Sep par defaut = TAB	Lecture d'un champ (Index=[1..x]) de BLOB de travail (CR)	Result=Texte,
BLOBReadFieldToBlob("BlobDest";Index[;Sep])	Lecture d'un champ (Index=[1..x]) de BLOB de travail dans BlobDest ,	Result=True=Ok (Sep par defaut = CR)
BLOBExtract(StartPosition[;Len])	Retire une partie du BLOB de travail , si pas Len, alors Len=(Size-StartPosition),	Result=True=Ok
BLOBFind(StrValue[;StartPosition[;CaseSensitif]])	Recherche la position d'une valeur dans BLOB de travaille ,	Result=Position, Result=-1= pas trouvé
BLOBFill(StrFill[;Count[;StartPosition]])	Remplit le BLOB de travail X fois de la valeur StrFill,	Result=True=Ok
BLOBCRC([CRC])	Calcul un CRC au BLOB (CRC= 8/16/32, avec 32 par défaut)	Result=Valeur du calcul
BLOBAddCRC([CRC])	Ajoute un CRC au BLOB (CRC= 8/16/32, avec 32 par défaut)	Result=True = Ok
BLOBTestCRC([CRC])	Test et retire un CRC au BLOB (CRC= 8/16/32, avec 32 par défaut)	Result=True = Ok

Les fonctions de traitement des chaînes

Fonction	Signification	Type du Résultat	Exemple
StrLower(String)	Convertit la chaîne en Argument en lettres minuscules	Chaîne	StrLower("WIT") = wit
StrUpper(String)	Convertit la chaîne en Argument en lettres majuscules	Chaîne	StrUpper("wit") = WIT
StrFind(SubString;String),	Recherche la sous-chaîne *SubString* dans la chaîne *String*	Booléen	StrFind("T" ; "WIT")=True
StrPos(SubString;String)	Retourne la position de la sous-chaîne *SubString* dans la chaîne *String*. Retourne 0 si non trouvé.	Entier	StrPos("T" ; "WIT")=3
StrCopy(String;StartIndex[;NbChar])	Extrait *NbChar* caractères à partir du *StartIndex* de la chaîne *String*.	Chaîne	StrCopy ("Formation" ;2 ;3)="orm"
StrField(String;FieldIndex[;SepChar])	Extrait le champ *FieldIndex* de la chaîne *String* dans laquelle le séparateur est *SepChar*.	Chaîne	StrField ("Toto;Tata;Titi";2 ;";")= "Tata"
StrFieldCount(String[;SepChar])	Retourne le nombre de champs de la chaîne *String* dans laquelle le séparateur est *SepChar*.	Entier	StrFieldCount ("Toto;Tata;Titi";";")= 3
StrRec(Field1;Field2~~FieldX)	Constitue une chaîne de caractère avec les différents arguments avec le séparateur Tabulation (09h).	Chaîne	
StrLength(String)	Retourne la longueur de la chaine *String*	Entier	
StrDecimal(Value[;NbDecimal])	Force le nombre de décimale de *Value* à *NbDecimal*. (Bourrage avec des 0)	Chaîne	StrDecimal(4,3 ;4)= "4,3000"
StrDigit(Value[;NbDigit])	Force le nombre de Digit de *Value* à *NbDigit* (Bourrage avec des 0)	Chaîne	StrDigit(23 ;4) = "0023"
StrVisible(String)	Supprime les caractères invisibles de la chaîne *String*	Chaîne	
StrLabel(String)	Modifie la chaîne *String* pour quelle puisse être utilisée en tant que Label d'un Objet	Chaîne	
StrPatchChar(String;Char;CharPatch)	Remplace le caractère *Char* de la chaîne *String* par le caractère *CharPatch*.	Chaîne	StrPatchChar ("TOTO";"O";"A") = "TATA"
StrDeleteChar(String[;Left[;Body [;Right[;Char]]]])	Supprime le caractère *Char* de la chaîne *String* à gauche, au centre, à droite. Mettre 1 pour supprimer ou 0 pour ne pas supprimer.	Chaîne	StrDeleteChar("AATITIAATOTOAA" ; 1 ;0 ;1 ; "A")= "TITIAATOTO"
StrHexaFromValue(Value;DataType)	Convertit *Value* en chaîne de caractère selon le *DataType*	Chaîne	StrHexaFromValue (10 ;#String)= "3130"
StrHexaToValue(String;DataType)	Convertit la chaîne *String* selon le *DataType*	Datatype	StrHexaToValue ("10";#long)=16
Char(NumChar)	Retourne le caractère Ascii de *NumChar*	Chaîne	Char(67)= "C"
CharNum(Char)	Retourne la valeur du caractère Ascii *Char*	Entier	CharNum("C")=67
Intl("Fra|Eng|Deu|Ita|Span|Pol")	Retourne la bonne chaîne en fonction de la langue paramétrée.	Chaîne	

Les fonctions de traitement des dates

Fonction	Signification	Type du Résultat	Exemple
Year(WitTime)	Retourne l'année du *WitTime*	Chaîne	Year(Clock)=7
Month(WitTime)	Retourne le mois du *WitTime*	Chaîne	
Day(WitTime)	Retourne le Jour du *WitTime*	Chaîne	
Hour(WitTime)	Retourne l'heure du *WitTime*	Chaîne	
Minute(WitTime)	Retourne les minutes du *WitTime*	Chaîne	
Second(WitTime)	Retourne les secondes du *WitTime*	Chaîne	
DayOfYear(WitTime)	Quantième de l'année	Chaîne	DayOfWeek(Clock)=304
DayOfWeek(WitTime)	Jour de la semaine (1=Lundi..7=Dimanche)	Chaîne	
WeekOfYear(WitTime)	Numéro de semaine	Chaîne	
MonthDayCount(WitTime)	Nombre de jours dans le mois	Chaîne	
Date(WitTime[;Short])	Date du *WitTime* au format JJ/MM/AAAA ou JJ/MM/AA si *short* est renseigné	Chaîne	Date(Clock ;1)= "31/10/07"
Time(WitTime)	Heure du *WitTime* au format HH :MM :SS	Chaîne	
DateTime(WitTime)	Date et Heure du *WitTime* au format JJ/MM/AAAA HH :MM :SS	Chaîne	
Clock	Nombre de secondes écoulées depuis le 1er Janvier 2000	Entier	
ClockGMT	Nombre de secondes écoulées depuis le 1er Janvier 2000 en Heure GMT	Entier	
ClockSet(WitTime[;DeltaSec]) WitTime=Heure en donnée local à écrire DeltaSec=Décalage minimum pour autoriser l'écriture	Ecriture de l'heure	Booléen	
ClockGMTSet(WitTime[;DeltaSec]) WitTime=Heure en donnée local à écrire DeltaSec=Décalage minimum pour autoriser l'écriture	Ecriture de l'heure GMT	Booléen	
GMTToLocal(WitTime)	Donne l'heure locale à partir de l'heure GMT	WitTime	
LocalToGMT(WitTime)	Donne l'heure GMT à partir de l'heure Locale	WitTime	
WitTimeToYMDHMS(WitTime)	Convertit le *WitTime* en une chaîne au format AAAAMMJJHHMMSS	Chaîne	WitTimeToYMDHMS (Clock)= « 20071031105613 »
WitTimeOfYMDHMS(« YMDHMS »)	Convertit la chaîne AAAAMMJJHHMMSS en un *WitTime*	Entier	
WitTimeOfDate(« DD/MM/YYYY »)	Convertit la chaîne « DD/MM/YYYY » en un *WitTime*	Entier	
WitTimeOfTime(« HH :MM :SS »)	Convertit la chaîne « HH :MM :SS » en un *WitTime*	Entier	
WitTimeOfDateTime ("DD/MM/YYYY HH:MM:SS")	Convertit la chaîne « DD/MM/YYYY HH :MM :SS » en un *WitTime*	Entier	

Les fonctions de conversion

Fonction	Signification	Type du Résultat	Exemple
Boolean(Value)	Convertit la Value en booléen	Booléen	
Small(Value)	Convertit la Value en entier 8 bits	Entier 8 Bits	
Short(Value)	Convertit la Value en entier 16 bits	Entier 16 Bits	
Long(Value)	Convertit la Value en entier 32 bits	Entier 32 Bits	Long(23,4)=23
Single(Value)	Convertit la Value en réel simple précision	Réel simple précision	Single(23,4) = 23,39999962
Double(Value)	Convertit la Value en réel double précision	Réel double précision	Double(23,4)=23,4
String(Value)	Convertit la Value en chaîne	Chaîne	

Les fonctions de traitement des booléens

Fonction	Signification	Type du Résultat	Exemple
Not(Value)	Retourne le complément de *Value*	Booléen	
BitTest(Integer ;NoBit)	Retourne la valeur du Bit *NoBit* (0-31) de l'entier *Integer*	Booléen	BitTest(131072 ;17)=True
BitSet(Integer ;NoBit)	Force à 1 la valeur du Bit *NoBit* de l'entier *Integer*	Entier	
BitClear(Integer ;NoBit)	Force à 0 la valeur du Bit *NoBit* de l'entier *Integer*	Entier	
BitCpl(Integer ;NoBit)	Complémente le Bit *NoBit* de l'entier *Integer*	Entier	
BitShiftL(Integer ;NoBit)	Décalage à Gauche de l'entier *Integer* du nombre de Bit *NoBit*	Entier	BitShiftL(16 ;1)=32
BitShiftR(Integer ;NoBit)	Décalage à Droite de l'entier *Integer* du nombre de Bit *NoBit*	Entier	
Band(Integer1 ;Integer2~~IntegerX)	Fonction ET entre tous les Entiers Arguments	Entier	Band(17 ;19 ;21)=17
Bor(Integer1 ;Integer2~~IntegerX)	Fonction OU entre tous les Entiers Arguments	Entier	
BXOr(Integer1 ;Integer2~~IntegerX)	Fonction OU EXCLUSIF entre tous les Entiers Arguments	Entier	

Les fonctions diverses

Fonction	Signification	Type du Résultat	Exemple
Index(Value ; Arg0 ; Arg1~~ArgX)	Retourne l'index (de 1 à N) pour lequel *Value* = *Argument*. Retourne 0 si non trouvé	Entier	Index(:easy.RESS.R00003.R00002.Output;13 ;14 ;15 ;16 ;17)=3 si :easy.RESS.R00003.R00002.Output =15
Select(Index ; Arg0 ; Arg1~~ArgX)	Sélectionne l'*Argument* en fonction de l'*Index*. Si *Index*=0, retourne *Arg0*, si Index=1 retourne *Arg1*…..	Chaîne	select(:easy.RESS.R00003.R00002.Output-14;6;13;23,4;16;6)=13 si :easy.RESS.R00003.R00002.Output =15

Les fonctions de traitement des ensembles

Fonction	Signification	Résultat
SETScanInit	Sélectionne un SET de travaille, et initialise le Scan	(True=Ok)
SETScanNext	Rend le chemin du prochain item dans l'Ensemble de travaille	(True=Ok)
SETClear	Vide l'Ensemble de travaille	(True=Ok)
SETAddObject("Object")	Ajoute Object à l'Ensemble de travaille	(True=Ok)
SETSubObject("Object")	Retire Object de l'Ensemble de travaille	(True=Ok)
SETExistsObject("Object")	Test si Object fait partie de l'Ensemble de travaille	(True=Ok)
SETCountObject	Rend le nombre d'Object dans l'Ensemble de travaille	(True=Ok)

Chaque fonction SET doit savoir sur quel ensemble travailler.
Pour cela il est nécessaire d'écrire en tête de script:

SETScanInit("SET1")

SET1 représente le label de l'ensemble défini dans \Paramétrage\Ensembles

Exemple:

```
SETScanInit("SET1")
  repeat
    MyStr = SETScanNext
    if (StrLength(MyStr)>0) then
      @(MyStr&&".Zone") = ":System.Attribute.Zone.Z_2"
    end
  until (StrLength(MyStr)=0)
```

Les fonctions logarithmiques

Fonction	Signification	Type du Résultat
Log(Value[;Base])	Retourne le Logarithme à Base *Base* de *Value*	Réel
Log2(Value)	Retourne le Logarithme à Base 2 de *Value*	Réel
Log10(Value)	Retourne le Logarithme à Base 10 de *Value*	Réel

Sur événement

Fonction	Signification	Exemple
OnInit	Permet de faire un traitement au démarrage du produit ou à la compilation du script	if OnInit then .ClearFTP = 56 end

Les fonctions de traitement des évènements

Fonction	Signification	Résultat
EventNew("Status"[;Broadcast[;"Kind"]]) Status: états Broadcast : 1=diffusion, 0= non diffusion Kind = type: A => apparition D => disparition # => système Autres caractères => one shot	Création d'un nouvel évènement	Result=EventID (0=nil)
EventYoung(["ObjectOwner"]) ObjectOwner = objet propriétaire	Donne l'évènement le plus récent	Result=EventID (0=nil)
EventPrev(EventID[;"ObjectOwner"])	Donne l'évènement précèdent	Result=EventID (0=nil)
EventNext(EventID[;"ObjectOwner"])	Donne l'évènement suivant	Result=EventID (0=nil)
EventOld(["ObjectOwner"])	Donne l'évènement le plus ancien	Result=EventID (0=nil)
EventExport(EventID[;"Format"]) Format : ^1 ID de l'évènement ^2 Date d'origine ^3 Libellé du site d'origine ^4 Libellé de la source ^5 Etat ^6 Type de l'évènement ^7 URL de la source ^8 Libellé du site serveur ^T Tabulation ^R Retour Chariot	Exporte l'évènement (Format par Défaut = "ID+TAB +DateOrg+TAB+Site+TAB+Caption +TAB+State+TAB +Kind")	Result=Chaine Exemple : Format: ^2^R<<^8>>^RRess: ^4^REtat: ^5 Résultat: 21/05/2008 17:05:48 <<GDF NOGARO>> Ress: Ress0001 Etat: ON
EventDateRec(EventID)	Date de l'enregistre de l'évènement	Result=TimeLong (FEventDateRec)
EventDateOrg(EventID)	Date de la création de l'évènement	Result=TimeLong (FEventDateOrg)
EventURL(EventID)	Donne l'url de la ressource qui a générée l'évènement	Result=Chaine (FEventURL)
EventSite(EventID)	Donne le nom du site d'où provient l'évènement	Result=Chaine (FEventSite)
EventCaption(EventID)	Donne le libellé de la ressource qui a générée l'évènement	Result=Chaine (FEventCaption

EventState(EventID)	Donne le statut de l'évènement	Result=Chaine (FEventState)
EventKind(EventID)	Donne le type de l'évènement	Result=Chaine (FEventKind) "?" = pas de type spécifique "." = évènement one shot "A"= évènement sur apparition "D"= évènement sur disparition "#" = évènement système
EventBroadcast(EventID)	Diffuse l'évènement	Result=Boolean (FEventBroadcast)
EventKill(EventID)	Efface l'événement	Result=Boolean (True=Ok)
EventExists(EventID)	Test si l'évènement existe	Result=Boolean (True=Existe)
EventAddJointFile(EventID;"FileName"[;"Object"])	Création d'un fichier joint concernant l'évènement (Voir Exemple)	Result=True si Ok
EventKillJointFile(EventID)	Effacer le fichier joint de l'évènement	Result=True si il y avait au moins 1 fichier

Exemple de pièce jointe :

```
MyOk = EventAddJointFile(MyId;"Ma Piece Jointe";":WEB.IMG.bureau~zip")
                                    ou
BLOB(":WEB.IMG.bureau~zip")
MyOk = EventAddJointFile(MyId;" Ma Piece Jointe")
```

Les fonctions de traitement des fichiers

Fonction	Signification
FileExist("FileName")	Test si le fichier *FileName* existe
FileDate("FileName")	Retourne la date du fichier *FileName*
FileSize("FileName")	Retourne la taille du fichier *FileName*
FileCopy("FileNameSrc";"FileNameDest")	Copie le fichier *FileNameSrc* dans le fichier *FileNameDest*
FileAppend("FileNameSrc";"FileNameDest")	Copie le fichier *FileNameSrc* à la fin du fichier *FileNameDest*
FileClear("FileName")	Clear le fichier *FileName*
FileDelete("FileName")	Supprime le fichier *FileName*
FileRename("FileNameOld";"FileNameNew")	Renomme le fichier *FileName*

FileWrite("FileNameDest";"StringValue")	Ecrit la Chaîne *StringValue* dans le fichier *FileNameDest*
FileToObject("FileNameSrc";"ObjectDest"[;Append [;Start[;Size]]]) Append= Ajouter : 0 ou 1 Start=commencer : nb de caractères à partir duquel on commence Size =Taille : nombre de caractères total	Ecrit le FileNameScr dans l'ObjectSrc
FileFromObject("ObjectSrc";"FileNameDest"[;App end])	Ecrit l'ObjectSrc dans le FileNameScr
DirectoryGet	Retourne le nom du dossier de travail
DirectorySet("DirectoryPath")	Force le dossier *DirectoryPath* comme dossier de travail

Fonctions de traitement des tableaux

Fonction	Signification	Type du Résultat	Exemple
ChildCreate("."; "Nom du tableau";41)	Créé un tableau		ChildCreate(".";"Valeurs ";41)
TABLE("Object")	Sélectionne Le Tableau de travaille, (BLOB)	Booléen	Result=True=Ok
TABLEDim(XCount;YCount)	Redéfinit la taille du Tableau XMax e@syPilot = 1000 XMax e@sy/REDY = 100 YMax e@syPilot = 10000 YMax e@sy/REDY = 1000	Booléen	Result=True=Ok
TABLEClear	Efface le contenu du tableau	Booléen	Result=True=Ok
TABLESize	Compacte la table et Rend sa taille en mémoire.	Entier	Result=Taille en Octet
CELLValid(X;Y)	Test si une cellule du Tableau est valide.	Booléen	Result=True=Ok
CELLClear(X1;Y1[;X2[;Y2]])	Efface une Zone de cellules du Tableau, le format de la cellule devient non valide.	Booléen	Result=True=Ok
CELLFill(X1;Y1;X2;Y2;Value)	Ecrit une Zone de cellules du Tableau, Type de Value conditionne le format de la cellule.	Booléen	Result=True=Ok

CELLRead(X;Y[;Formula])	Lit une cellule du Tableau, Type de Result suivant format de la cellule,	Booléen Chaine Entier	Result=Valeur (si Formula=True, alors relit la formule au lieu de la valeur)
CELLWrite(X;Y;Value)	Ecrit une cellule du Tableau, Type de Value conditionne le format de la cellule,	Booléen	Result=True=Ok

*Exemple de création d'un tableau nommé **Valeurs**, pouvant contenir 60 valeurs :*

```
ChildCreate(".";"Valeurs";41)
  TABLE(".Valeurs")
  TABLEDim(60;1)
```

Ressource

Fonction	Signification	Type du Résultat	Exemple
RessStatus(« Objet »)	Retourne l'état de l'Objet	Chaine	RessStatus(":easy.RESS.ExtenBUS") Result= Déconnecté

Les fonctions de traitement des objets

Fonction	Signification	Type du Résultat	Exemple
Self	Chemin complet de l'objet lui-même	Chaîne	
ParentLabel(« Object »)	Label du Parent de l'*Object*	Chaîne	ParentLabel (":easy.RESS.R00003.R00006") =R00003
ParentPath(« Object »)	Chemin complet du Parent de l'*Object*	Chaîne	ParentPath (":easy.RESS.R00003.R00006") = :easy.RESS.R00003
ObjectLabel(« Object »)	Label de l'*Object*	Chaîne	ObjectLabel (":easy.RESS.R00003.R00006") =R00006
ObjectPath(« Object »)	Chemin complet de l'*Object*	Chaîne	ObjectPath (":easy.RESS.R00003.R00006") = :easy.RESS.R00006
ObjectURL(« Objet »)	Donne l'URL l'*Object*	Chaine	ObjectURL(":easy.RESS.R00003.R00007") =/easy/RESS/R00003/R00007
ObjectCaption(« Objet »)	Donne le libellé de l'*Object*	Chaine	ObjectURL(":easy.RESS.R00003.R00007") =Température extérieure
ObjectExist(« Objet »)	Renseigne si l'objet existe	Booléen	If ObjectExist(":easy.RESS.R00003.R00007") then
ObjectClass("Object")	Classe de l'*Object*	Chaîne	ObjectClass (":easy.RESS.R00003.R00006") =559
ObjectChange("Object")	Simule le changement de l'*Object* (non utilisé)		
ChildCreate ("Object";"Label"[;Class])	Permet de créer un objet de classe *Class* de Nom *Label* enfant de l'*Object* parent	Chaîne	
ChildCount("Object";Class)	Retourne le nombre d'enfant de classe *Class* enfants de l'*object* parent	Chaîne	
ChildLabel("Object";Index)	Retourne le label de l'enfant du parent *Object* dont le numéro d'ordre est *Index*. Le premier Objet enfant a 1 pour *Index.*	Chaîne	ChildLabel (":easy.RESS.R00003";3) =R00006 La ressource R00006 est le 3e enfant de R00003.
ChildPath("Object";Index)	Retourne le chemin complet de l'enfant du parent *Object* dont le numéro d'ordre est *Index*. Le premier Objet enfant a 1 pour *Index.*	Chaîne	
ChildLabelList ("Object";Class;"BlobDest")	Ecrit dans *BlobDest* la liste des labels des Enfants de classe *Class* dont le parent est *Object*	Chaîne	
ChildPathList (Object";Class;"BlobDest")	Ecrit dans *BlobDest* la liste des chemins des Enfants de classe *Class* dont le parent est *Object*	Chaîne	
DescendantPathList ("Object";Class;"BlobDest"')	Ecrit dans *BlobDest* la liste des chemins des Descendants de classe *Class* dont le parent est *Object*	Chaîne	
DescendantSetGroupe ("Object";NumGroupe;Value)		Chaîne	
ObjectSetGroupe(« Object » ;NumGroupe ;Value)	Permet d'affecter ou désaffecter un groupe d'un objet		ObjectSetGroupe(":System.User.U00002.GroupJrnl";3;1)
ObjectGetGroupe(« Object » ;NumGroupe)	Permet de savoir l'appartenance d'un groupe d'un objet	Booléen	ObjectGetGroupe(":System.User.U00002.GroupJrnl";3)
Value("Object")	Retourne la valeur de l'*Object*	Type de l'Objet	Value(":easy."&& "RESS.R00003.R00006.Output") =19,6

Fonctions de traitement des traces

Fonction	Signification
TraceFnctExportBlob("TraceFnct"[;WitTimeStart[;WitTimeEnd[;YMDHMS]]]) WitTimeStart=Date de début de l'export WitTimeEnd=Date de fin de l'export YMDHMS= False = JJ/MM/AAAA HH:MM:SS True = AAAAMMJJHHMMSS	Exporte les pas d'une fonction trace dans un BLOB entre 2 dates. TraceFnctExportBlob(:easy.RESS.ExtenBUS.FNCT.F0001;Clock-3600;Clock) Result=Nombre de Pas exporté

Fonction trigonométriques

Fonction	Signification	Type du Résultat
RadFromDeg(Value)	Convertit la *Value* de Deg en Radian	Entier
RadToDeg(Value)	Convertit la *Value* de Radian en Degré	Entier
RadFromGrad(Value)	Convertit la *Value* de Grade en Radian	Entier
RadToGrad(Value)	Convertit la *Value* de Radian en Grade	Entier
Sin(Value)	Retourne le Sinus de *Value*	Réel
Cos(Value)	Retourne le Cosinus de *Value*	Réel
Tan(Value)	Retourne la Tangente de *Value*	Réel
ArcSin(Value)	Retourne l'Arc sinus de *Value*	Réel
ArcCos(Value)	Retourne l'Arc cosinus de *Value*	Réel
ArcTan(Value)	Retourne l'Arc tangente de *Value*	Réel
HSin(Value)	Retourne le Sinus Hyperbolique de *Value*	Réel
HCos(Value)	Retourne le Cosinus Hyperbolique de *Value*	Réel
HTan(Value)	Retourne la Tangente Hyperbolique de *Value*	Réel
HArcSin(Value)	Retourne l'Arcsinus Hyperbolique de *Value*	Réel
HArcCos(Value)	Retourne l'Arccosinus Hyperbolique de *Value*	Réel
HArcTan(Value)	Retourne l'Arctangente Hyperbolique de *Value*	Réel
CoTan(Value)	Retourne la Cotangente de *Value*	Réel
Hypot(ValueX;ValueY)	Retourne l'Hypoténuse des *ValueX*, *ValueY*= SQR(X^2+Y^2)	Réel

Fonction WEB

Fonction	Signification	Type du Résultat	Exemple
WEBFileCreate("ObjectDir";"FileName")	Créer un fichier WEB enfant de ObjectDir,	Chaine	Result=NodPath du fichier
WEBFolderCreate("ObjectDir";"FolderName")	Créer un dossier WEB enfant de ObjectDir,	Chaine	Result=NodPath du fichier

WEBFileFolderExists("ObjectDir";"FileFolderName")	Test si un fichier/folder WEB existe enfant de ObjectDir,	Booléen	Result=True si existe
WEBFileFolderFind("ObjectDir";"FileFolderName")	Recerche un fichier/folder WEB enfant de ObjectDir,	Chaine	Result=NodPath du fichier/folder
WEBFileFolderDelete("ObjectDir";"FileFolderName")	Détruit un fichier/folder WEB enfant de ObjectDir	Booléen	Result=True si fichier trouvé
WEBFileFolderRename("ObjectDir";"OldName";"NewName")	Renomme un fichier/folder WEB enfant de ObjectDir	Booléen	Result=True si fichier renommé

Fonction WOD

Fonction	Signification
WODRename("ObjectSrc";"NewName")	Renomme le WOD *ObjectSrc* avec le nom *NewName*
WODExist("ObjectSrc")	Teste si le WOD *ObjectSrc* existe
WODDelete("ObjectSrc")	Supprime le WOD *ObjectSrc*
WODDuplicate("ObjectSrc";"NewName")	Duplique le WOD *ObjectSrc* avec le nom *NewName*
WODPublic("ObjectSrc";#True|#False[;#True])	Rend Public le WOD *ObjectSrc* *Le 3° Argument indique si on veut répercuter l'affection aux enfants de ObjectSrc*
WODCopyToBLOB("ObjectSrc";"BlobDest")	Copy le WOD *ObjectSrc* dans le BLOB *BLOBDest*
WODCopyValueToBLOB("ObjectSrc";"BlobDest")	Copy la valeur du WOD *ObjectSrc* dans le BLOB *BLOBDest*
WODPasteFromBLOB("BlobSrc";"ObjectDest")	Colle le BLOB *BLOBSrc* dans l'objet *ObjectDest*
WODAppendFromBLOB("BlobSrc";"ObjectDest"[;"Name"])	Ajoute le BLOB *BLOBSrc* dans l'objet *ObjectDest.* Le nom de l'objet crée est *Name.*
WODCopyToFile("ObjectSrc";"FileNameDest")	Copy le WOD *ObjectSrc* dans le fichier *FileNameDest*
WODCopyValueToFile("ObjectSrc";"FileNameDest")	Copy la valeur du WOD *ObjectSrc* dans le fichier *FileNameDest*
WODPasteFromFile("FileNameSrc";"ObjectDest")	Colle le Fichier *FileNameDest* dans l'objet *ObjectDest*
WODAppendFromFile("FileNameSrc";"ObjectDest"[;"Name"])	Ajoute le Fichier *FileNameDest* dans l'objet *ObjectDest.* Le nom de l'objet crée est *Name.*

Fonction JSON

Fonction	Signification
JSONParseBlob("BlobSrc"[;Position[;Len]])	> Argument "BlobSrc", chemin vers le BLOB contenant les données brutes JSON à analyser > Argument optionnel "Position", position de lecture (0 à "BLOBSize" - 1) au sein du Blob "BlobSrc" > Argument optionnel "Len", nombre de caractères à lire (au maximum "BLOBSize") au sein du Blob "BlobSrc"

	> Retourne l'index (1 à N où N vaut 10) de la structure JSON ajoutée dans la mémoire tampon, valeur négative si erreur. *Taille du BLOB source <= 2 Mo maximum*
JSONClearCache(Index)	> Argument "Index", index de la structure JSON dans la mémoire tampon (si l'index = 0, on vide la mémoire tampon > Retourne une valeur booléenne indiquant si la libération de la structure JSON choisie (via l'index) a été faite
JSONCountItems(Index[;JsonPath[;Deeply]])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument optionnel "JsonPath", chemin d'accès à l'élément JSON dans la structure choisie (via l'index) > Argument optionnel "Deeply", si présent et à vrai, on décompte tous les éléments présents à partir du chemin d'accès (si il n'est pas renseigné ou à faux, on décompte seulement les éléments de premier niveau, pas dans la profondeur) > Retourne le nombre d'éléments (de premier niveau ou de toute la profondeur) au sein de la structure choisie (via l'index), valeur négative si erreur
JSONItemExist(Index[;JsonPath])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument optionnel "JsonPath", chemin d'accès (servant à la recherche) à l'élément JSON dans la structure choisie (via l'index) > Indique si l'élément JSON recherché existe au sein de la structure choisie (via l'index), valeur négative si erreur
JSONItemType(Index[;JsonPath])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument optionnel "JsonPath", chemin d'accès à l'élément JSON dans la structure choisie (via l'index) > Retourne un entier représentant la nature de l'élément JSON présent dans la structure choisie (via l'index), valeur négative si erreur > Types possibles : Inconnu = 0, Nombre = 1, Chaîne = 2, Booléen = 3, Nul = 4, Tableau = 5 et Objet = 6
JSONItemValue(Index[;JsonPath])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument optionnel "JsonPath", chemin d'accès à l'élément JSON dans la structure choisie (via l'index) > Retourne sous forme d'une chaîne la valeur de l'élément JSON présent dans la structure choisie (via l'index), si erreur ou introuvable => chaîne vide

JSONCopyItem(Index[;JsonPath])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument optionnel "JsonPath", chemin d'accès à l'élément JSON (qui sera copié) dans la structure choisie (via l'index) > Retourne l'index de la copie de l'élément JSON présent dans la structure choisie (via l'index), valeur négative si erreur
JSONCopyItemToBlob(Index;"BlobDest"[;JsonPath])	> Argument "Index", index de la structure JSON dans la mémoire tampon > Argument "BlobDest", chemin vers le BLOB qui contiendra les données JSON brutes copiées > Argument optionnel "JsonPath", chemin d'accès à l'élément JSON (qui sera copié) dans la structure choisie (via l'index) > Retourne une valeur entière indiquant si la copie a été réalisée avec succès (valeur = 0), valeur négative si erreur

Code d'erreur

Dans le cas où une fonction retourne une erreur via une valeur négative, voici la signification de chaque valeur négative possible :

Code	Signification
0	Succès
-1	Erreur divers
-2	Mémoire insuffisante
-3	Blob demandé introuvable
-4	Taille du Blob (source) trop importante (> 2 Mo)
-5	Impossible d'analyser le flux JSON (probablement invalide)
-6	Impossible de trouver la structure (via l'index) dans la mémoire tampon
-7	Impossible d'ajouter une nouvelle structure dans la mémoire tampon
-8	Impossible de trouver l'élément JSON demandé (via "JsonPath")

Exemple d'un chemin d'accès JSON (JsonPath)

Exemple de la notion de "chemin d'accès JSON" :

```
{
 "firstName": "John",
 "lastName": "Smith",
 "isAlive": true,
 "age": 27,
 "address": {
  "streetAddress": "21 2nd Street",
  "city": "New York",
  "state": "NY",
```

```
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Mike",
    "Lisa",
    "Joe"
  ]
}
```


- Exemple 1 : Si on veut la valeur de la propriété "firstName" ➔ JsonPath = firstName
- Exemple 2 : Si on veut la valeur de la propriété "postalCode" de l'objet "address" ➔ JsonPath = address.postalCode
- Exemple 3 : Si on veut la valeur de la propriété "type" du premier objet au sein du tableau "phoneNumbers", JsonPath = phoneNumbers[0].type
- Exemple 4 : Si on veut la valeur de la propriété "number" du dernier objet au sein du tableau "phoneNumbers" ➔ JsonPath = phoneNumbers[1].number
- Exemple 5 : Si on veut la valeur du deuxième élément au sein du tableau "children" ➔ JsonPath = children[1]

Exemple d'analyse d'un BLOB

Analyse du contenu JSON présent dans le fichier "json_data" :


```
{
  "_id": "61e56d8d27199f024e550d93",
  "index": 0,
  "guid": "2676b86c-c73e-4f98-a479-698ba8729c1d",
  "isActive": true,
  "picture": "http://placehold.it/32x32",
  "age": 32,
  "eyeColor": "green",
  "name": "Garrison Hogan",
  "gender": "male",
  "company": "SKINSERVE",
  "email": "garrisonhogan@skinserve.com",
```

```
  "phone": "+1 (884) 572-3349",
  "address": "785 Onderdonk Avenue, Heil, Delaware, 4969",
  "tags": [
    "manager",
    "business",
    "electronic"
  ],
  "colleagues": [
    {
      "id": 0,
      "name": "Duran Stanley"
    },
    {
      "id": 1,
      "name": "Heidi Jackson"
    },
    {
      "id": 2,
      "name": "Rosa Little"
    }
  ],
  "registered": "2014-05-03T01:39:18 -02:00",
  "latitude": 10.316304,
  "longitude": -4.963989
}
```


Plus précisément, nous voulons récupérer les informations _is, name, isActive et les deux premiers éléments du tableau colleagues. Pour cela, nous allons écrire les instructions e@syScript suivantes :

```
var Blob MyJsonData
var Digital MyDoParsing
var Analog MyJsonStructIndex

var String MyIdentifier
var String MyName
var Digital MyIsActive
var String MyFirstColleague
var String MySecondColleague

// Au démarrage, on charge les données du fichier "json_data" situé dans le dossier ":WEB.IMG"
if OnInit then
        MyDoParsing = 0
  BLOB("MyJsonData")
  BLOBClear()
  // Si le fichier est bien chargé, alors on peut analyser les données JSON
        MyDoParsing = BLOBLoad(":WEB.IMG.test_json")
end

// On réalise l'analyse du Blob contenant les données JSON chargées
```

```
if MyDoParsing then
        MyDoParsing = 0
        MyJsonStructIndex = JSONParseBlob("MyJsonBlob")
end

// En effet, une analyse sans échec des données JSON provoque le stockage en mémoire d'une structure (située ici au premier index du tableau "mémoire tampon"). Cette structure permet d'accéder rapidement aux élément s JSON sans avoir à ré-analyser le contenu du Blob. Il est possible de stocker jusqu'à 10 structures (index de 1 à 10) au sein du tableau "mémoire tampon". Pour libérer de la place au sein du tableau "mémoire tampon", il fau dra utiliser la fonction "JSONClearCache(Index)", si l'index vaut 0, cela libère tous les emplacements.

// Si le Blob a bien été analysé, alors l'index doit normalement valoir 1
if (MyJsonStructIndex = 1) then

        // On vérifie la présence de la propriété JSON "_id"
        if (JSONItemExist(MyJsonStructIndex;"_id") = 0) then
                // On récupère la valeur au sein de la propriété JSON "_id"
    MyIdentifier = JSONItemValue(MyJsonStructIndex;"_id")
        end

        // On vérifie la présence de la propriété JSON "name"
        if (JSONItemExist(MyJsonStructIndex;"name") = 0) then
                // On récupère la valeur au sein de la propriété JSON "name"
    MyName = JSONItemValue(MyJsonStructIndex;"name")
        end

        // On vérifie la présence de la propriété JSON "isActive"
        if (JSONItemExist(MyJsonStructIndex;"isActive") = 0) then
                // Nous savons que la propriété "isActive" contient un valeur booléenne
                if (JSONItemType(MyJsonStructIndex;"isActive") = 3) then
                        // On récupère la valeur booléenne au sein de la propriété JSON "isActive"
        MyIsActive = Boolean(JSONItemValue(MyJsonStructIndex;"isActive"))
                end
        end

  // On vérifie la présence du tableau JSON "colleagues"
        if (JSONItemExist(MyJsonStructIndex;"colleagues") = 0) then
                // Nous souhaitons savoir si les deux premiers collègues sont présents
                if (JSONCountItems(MyJsonStructIndex;"colleagues") >= 2) then
                        // On récupère le nom du premier collègue
        MyFirstColleague = JSONItemValue(MyJsonStructIndex;"colleagues[0].name")
        // On récupère le nom du second collègue
        MySecondColleague = JSONItemValue(MyJsonStructIndex;"colleagues[1].name")
                end
        end

        // Nous avons terminé notre parcours des éléments JSON, on nettoie la structure présente dans la mém
oire tampon
```

```
        if JSONClearCache(MyJsonStructIndex) then
            // On ne retourne pas dans le bloc d'instructions > "if (MyJsonStructIndex = 1) then ... end"
            MyJsonStructIndex = 0
        end
end
```

NB : Nous aurions pu également libérer toutes les structures JSON en mémoire tampon dans le bloc d'instructions if MyDoParsing then ... end, voici ci-dessous la modification à apporter.

```
// On réalise l'analyse du Blob contenant les données JSON chargées
if MyDoParsing then
        MyDoParsing = 0
        // Suppression de toutes les anciennes structures
        JSONClearCache(0)
        MyJsonDataIndex = JSONParseBlob("MyJsonBlob")
end
```

Spécifique e@sy

Fonction FlashDisk

Fonction	Signification	Type du Résultat
FlashReady	Actif si la flach est prète a éxécuter une action	Booléen
FlashFree	Donne la taille disponible de la RAM en octet	Entier
FlashFileSize("FileName")	Donne la taille de *FileName* en octet	Entier
FlashFileToBLOB("FileName"))	Ecrit dans le Blob en cour le contenu du fichier *FileName*	Booléen
FlashFileFromBLOB("FileName")	Ecrit dans *FileName* le contenu du BLOB en cour	Booléen
FlashFileKill("FileName")	Efface *FileName*	Booléen
FlashFileExist("FileName")	Rend Vrais si le fichier *FileName* existe.	Booléen
FlashFileRename("FileNameOld";"File NameNew")	Renomme le fichier *FileNameOld* avec le nom *FileNameNew*	Booléen
FlashFileByIndex(Index)	Donne le nom du fichier numéroter *Index*	String

Les fonctions de traitement des objets

Fonction	Signification	Type du Résultat	Exemple
ObjectIDByPath("Object")	Fournit le numero ID de *Object*	Entier	ObjectIDByPath(":easy.RESS.R00002") = 1743
ObjectPathByID("ID")	Fournit le chemin corespondant a *ID*	Chaine	ObjectPathByID("1776") = ":easy.RESS.R00002.Output"

Les fonctions de traitement des chaînes

Fonction	Signification	Type du Résultat	Exemple
StrToSQLStr(String)	Formate la chaine *String* de caratères au format chaine SQL (Rajoute ')	Chaine	StrToSQLStr("Chaine d'initialisation") = "'Chaine d''initialisation'"

Spécifique e@sy-pilot

Les fonctions de traitement des fichiers

Fonction	Signification
WinExecute("ExeName";Arguments)	Exécute un exécutable Windows
DirectoryGet	Retourne le nom du dossier de travail
DirectorySet("DirectoryPath")	Force le dossier *DirectoryPath* comme dossier de travail

Fonctions de traitement des traces

Fonction	Signification
TraceBoolean("TraceName";Value[;WitTime])	Ecrit dans la Trace *TraceName* la *Value* Booléenne avec la Date *WitTime*
TraceReal("TraceName";Value[;WitTime])	Ecrit dans la Trace *TraceName* la *Value* Entière avec la Date *WitTime*
TraceString("TraceName";Value[;WitTime])	Ecrit dans la Trace *TraceName* la *Value* Chaîne avec la Date *WitTime*
TraceFirstDate("TraceName")	Retourne en WitTime la date du premier échantillon de la Trace *TraceName*
TraceNextDate("TraceName" ;WitTime)	Retourne en WitTime la date de l'échantillon suivant l'échantillon correspondant au paramètre WitTime de la Trace *TraceName*
TraceLastDate("TraceName")	Retourne en WitTime la date du dernier échantillon de la Trace *TraceName*
TraceDate("TraceName";WitTime)	Retourne en WitTime la date de l'échantillon la plus proche de *WitTime* de la Trace *TraceName*
TraceValue("TraceName";WitTime ;Value)	Permet de changer la valeur d'un pas déjà existant et ayant comme date WitTime de la Trace *TraceName*
TraceValueChange("TraceName";WitTime)	Retourne la valeur de l'échantillon le plus proche de *WitTime* de la Trace *TraceName*

TraceExport("TraceNameSrc";"FileNameDest"[;WitTimeStart[;WitTimeEnd[;Info[;Append[;EndChar]]]]])	Exporte la Trace *TraceNameSrc* dans le fichier *FileNameDest*
TraceFullExport("TraceNameSrc";"FileNameDest"[;WitTimeStart[;WitTimeEnd[;Info[;Append[;EndChar]]]]])	Exporte la Trace *TraceNameSrc* dans le fichier *FileNameDest.* Réalise une extrapolation sur les échantillons non présents dans le fichier.
TraceBooleanImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Booléen de la Trace *FileNameSrc* dans le fichier *TraceNameDest*
TraceRealImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Réel de la Trace *FileNameSrc* dans le fichier *TraceNameDest*
TraceStringImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Chaîne de la Trace *FileNameSrc* dans le fichier *TraceNameDest*
TraceAppend("TraceNameSrc";"TraceNameDest"[;WitTimeStart[;WitTimeEnd]])	Ajoute les échantillons du fichier *TraceNameSrc* à la fin du fichier *TraceNameDest*
TracePurge("TraceNameSrc"[;WitTimeStart[;WitTimeEnd]])	Réalise la purge de la Trace *TraceNameSrc*
TraceGetInfo("TraceName";[WitTimeStart[;WitTimeEnd]])	Lit les informations du fichier Trace *TraceName* JJ/MM/AAA HH:MM:SS(ValMin)|ValMin|ValMoy| JJ/MM/AAA HH:MM:SS(ValMax)|ValMax |NbStep
TraceFileName("TraceName")	Rend le chemin du fichier TRA par rapport au Path de la fonction Trace (TraceName)

Les fonctions de traitement SQL

Le paramètre TimeOut(Sec) correspond au délai Max de l'exécution du script SQL.

Fonction	Signification	Type du Résultat	Exemple
SQLSelect (SQLScript[;TimeOut])	Exécute un script SQL Select contenu dans une chaine	Chaine	SQLSelect("SELECT MAX(ID) FROM LISTE_SITES") Result= Résultat du SELECT donc la longueur <= 255. Les champs sont séparés par le caractère [Tab]
SQLSelectToTABLE (SQLScript[;TimeOut])	Exécute un script SQL Select contenu dans un BLOB et rempli une table(41) avec le résultat du select	Booléen	ChildCreate(".";"TableSQL";41) TABLE(".TableSQL") result = SQLSelectToTABLE("Select * from JRNL") CELLRead(4;1)
SQLSelectBLOBToStr([TimeOut])	Exécute un script SQL Select contenu dans un BLOB(40)	Chaine	ChildCreate(".";"BlobSQL";40) BLOB(".BlobSQL") BLOBWrite("Select * from JRNL") result = SQLSelectBLOBToStr ()

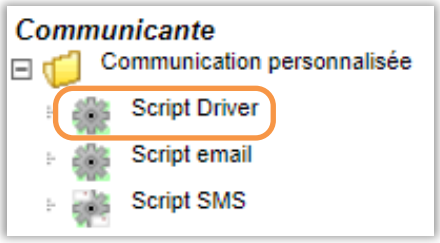
SQLSelectBLOBToTABLE([TimeOut])	Exécute un script SQL Select contenu dans un BLOB et rempli une table(41) avec le résultat du select	Booléen	ChildCreate(".";"BlobSQL";40) BLOB(".BlobSQL") BLOBWrite("Select * from JRNL") TABLE(".TableSQL") result = SQLSelectBLOBToTABLE()
SQLExec(SQLScript[;TimeOut])	Exécute un script SQL autre qu'un SELECT contenu dans une chaine	Booléen	SQLExec("TRUNCATE TABLE LISTE_SITES")
SQLExecBLOB([TimeOut])	Exécute un script SQL autre qu'un SELECT contenu dans un BLOB(40)	Booléen	ChildCreate(".";"BlobSQL";40) BLOB(".BlobSQL") BLOBWrite(" TRUNCATE TABLE LISTE_SITES ") SQLExecBLOB
SQLError	Retourne l'error SQL de la dernière fonction SQLSelect ou SQLExec exécutée	Chaine	Result= « Nom d'objet 'LISTE_SITES' non valide »

La ressource Script Driver

L'ULI permet de réaliser son propre protocole de communication.
Pour cela, il est nécessaire de créer une ressource Script Driver couplée avec une connexion utilisant le protocole **Driver Script**.
Cette ressource s'utilise comme une ressource **Script**, avec un dossier de fonctions supplémentaires.

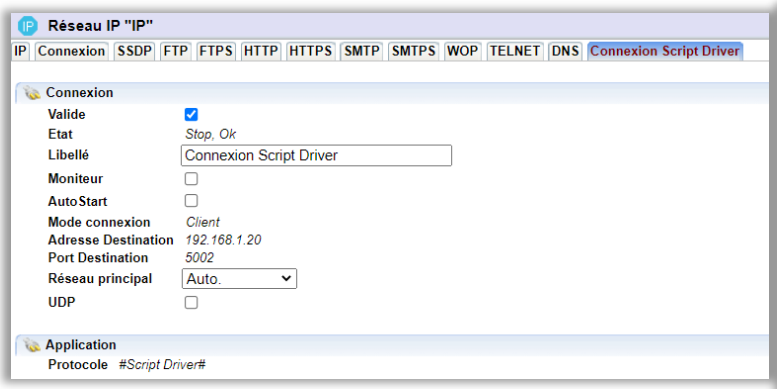
Paramétrage de la ressource

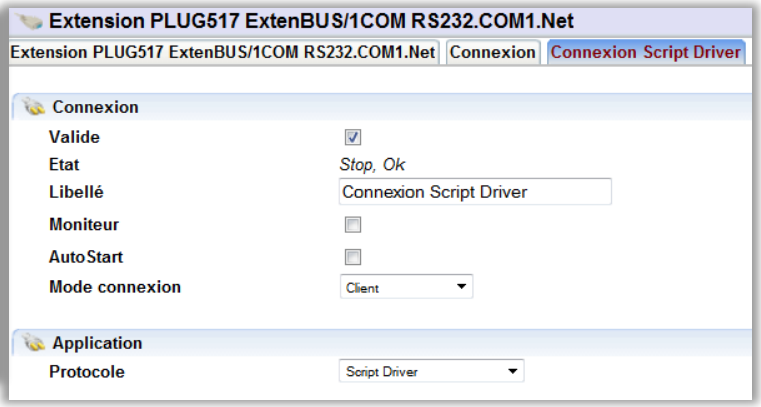
Etape 1 Ajouter une ressource Script Driver depuis le menu Paramétrage ► Ressource ► Ajouter une ressource ► Communication personnalisée ► Script Driver



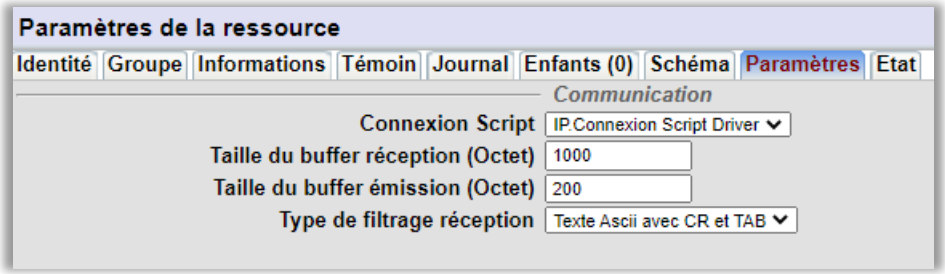
Etape 2 Créer la connexion (IP ou série) qui sera associé au Script, en lui appliquant le protocole **Script Driver**

- La connexion doit être **Valide**
- L'**Auto Start** doit être coché ou non selon le fonctionnement désiré.
- Le **Mode connexion** est en Client si on est à l'initiative de la communication, est en serveur si l'on attend des informations.
- Dans le cas d'une communication IP, définir le **Port destination**, et si l'on est maitre, l'**Adresse Destination**.





Etape 3 Lier la connexion au Script Driver et définir la taille des buffers de réception et d'émission.



Il est possible d'ajouter un filtre de réception (autorisation des trames texte au format **ASCII** contenant comme caractère de fin un **CR** ou un **TAB** *Char(13)*).

Fonctions

Fonction	Signification	Type du Résultat	Exemple
Connected	Indique si la Cnx est connectée	Booléen	Result=True Quand connexion en service
Disconnect	Demande de déconnexion de la Cnx	Booléen	Result=False Quand connexion en hors service
RxSize	Nombre de caractères dans le buffer de réception	Réel	
RxFree	Nombre de caractère libre dans le buffer de réception	Réel	Result=Taille Max – Taille utilisée
RxClear	Vide le buffer de réception	Booléen	
RxChar	Extrait le 1° caractère	Chaine	Result=1° carcatère
RxString	Extrait une chaine <= 255	Chaine	Chaine <= 255
RxLine	Extrait une chaine <= 255 ayant comme délimiteur CR	Chaine	Type de filtrage = CR Chaine sans le CR Sans Type de filtrage Chaine <= 255
RxBLOB	Extrait un BLOB ayant comme fin de trame CR	BLOB	
TxSize	Nombre de caractères dans le buffer d'émission	Réel	
TxFree	Nombre de caractère libre dans le buffer d'émission	Réel	Result=Taille Max – Taille utilisée
TxClear	Vide le buffer d'émission	Booléen	
TxString(String)	Rempli le buffer d'émission d'une *String*	Booléen	Emission de la trame Start+H03 TxString(«Start »&&Char(03))
TxLine(String)	Rempli le buffer d'émission d'une *String* et rajoute CR	Booléen	Emission de la trame Start+H03+H13 TxLine(«Start »&&Char(03))

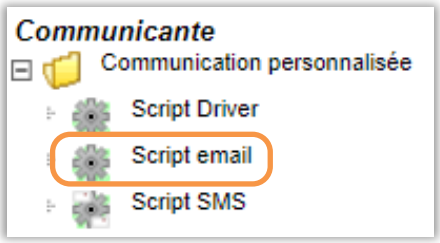
La ressource Script email

La ressource Script email permet de réceptionner les mails contenus dans un serveur de messagerie POP3 et de traiter leurs contenus. Cette ressource contient le dossier de fonction **Communication**.

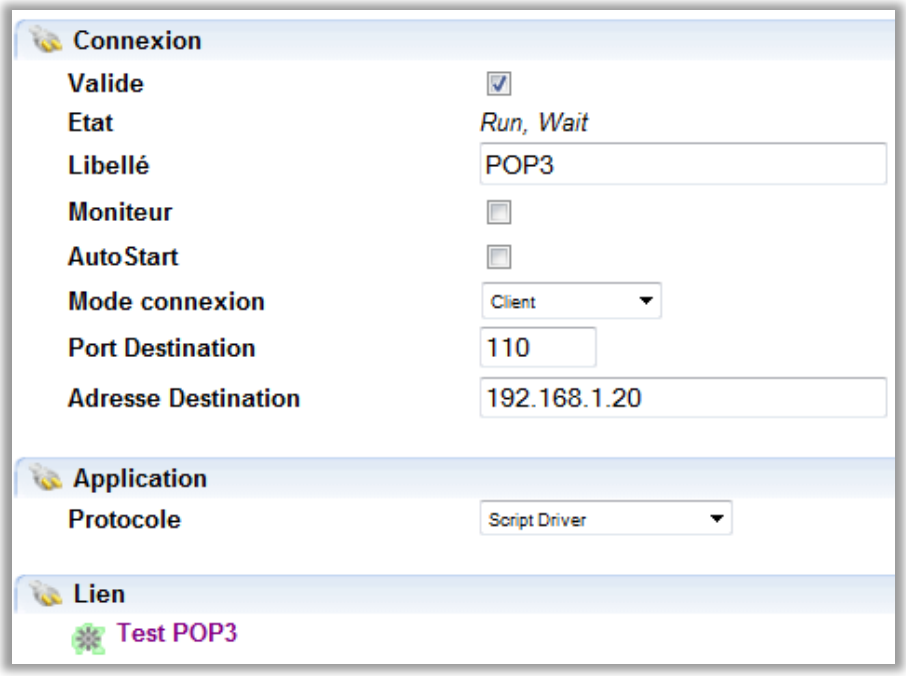
Le protocole **POP** (*Post Office Protocol*) permet d'aller récupérer son courrier sur un serveur distant (le serveur POP).

Paramétrage de la ressource

Etape 1 Ajouter une ressource Script email depuis le menu Paramétrage ▶ Ressource ▶ Ajouter une ressource ▶ Communication personnalisée ▶ Script email

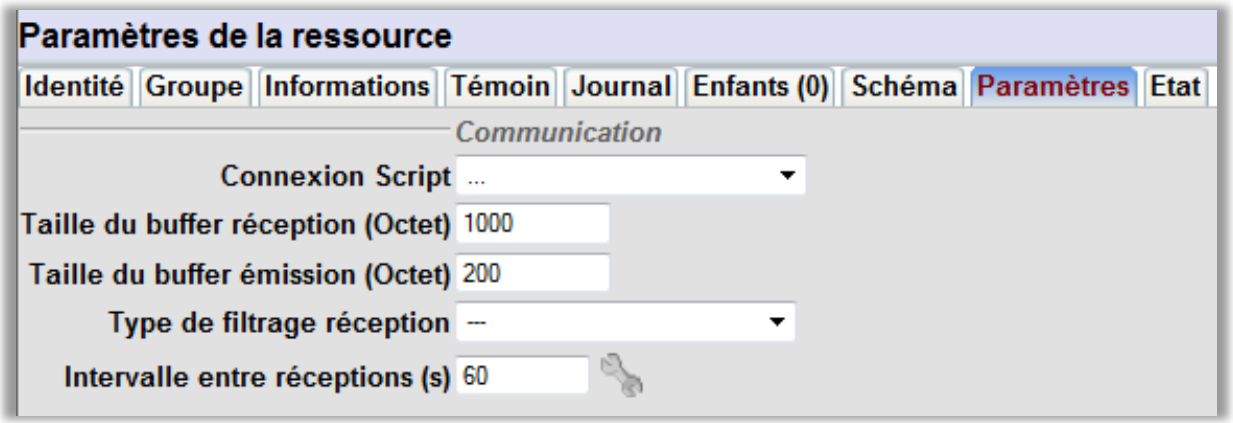


Etape 2 Créer la connexion IP associée au Script, en lui appliquant le protocole **Script Driver**



Mode de connexion : Client
Port de destination : Le port par défaut est110
Adresse de destination : Adresse à laquelle se trouve le serveur POP3 sur le réseau
Protocole : Script Driver

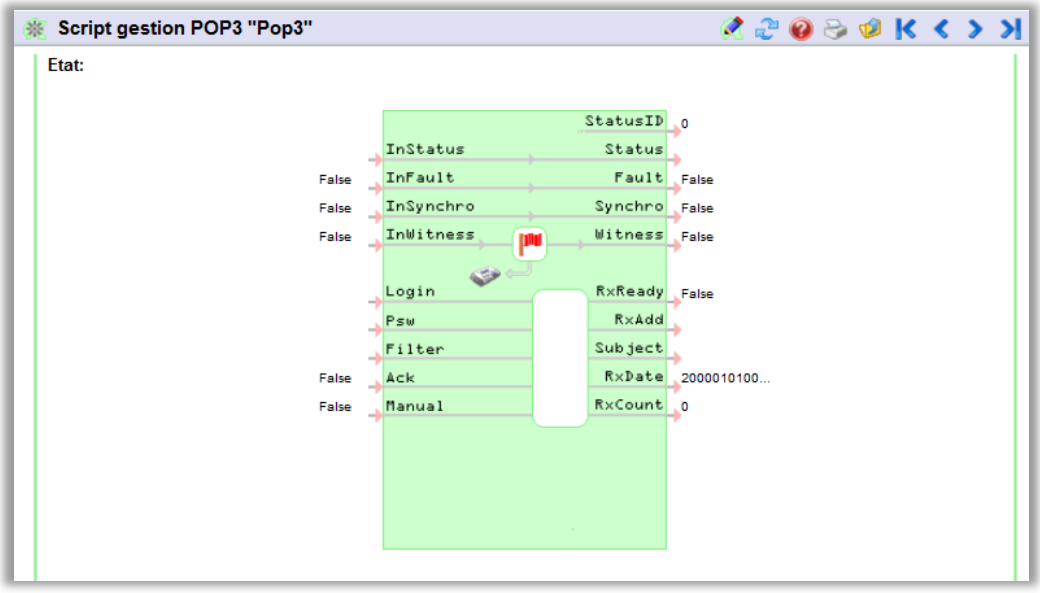
Etape 3 Lier la connexion au Script email et définir la taille des buffers de réception et d'émission.



Connexion Script : Permet de lier la ressource à la connexion POP3.
Taille du buffer de réception : Valeur par défaut 1000 Octets. *Non utilisé*
Taille du buffer d'émission : Valeur par défaut 200 Octets. *Non utilisé*
Type de filtrage réception : Sans filtrage ou bien avec « Cr+LF ».
Intervalle entre réceptions : Spécifie un temps entre lecture de 2 messages mail *(60 secondes par défaut)*.

Les e-mails doivent être envoyés au format « Texte brut ».

Liens de la ressource



Lors de la création de la ressource, des variables d'entrées/sorties sont déjà présentes :

Variables d'entrée

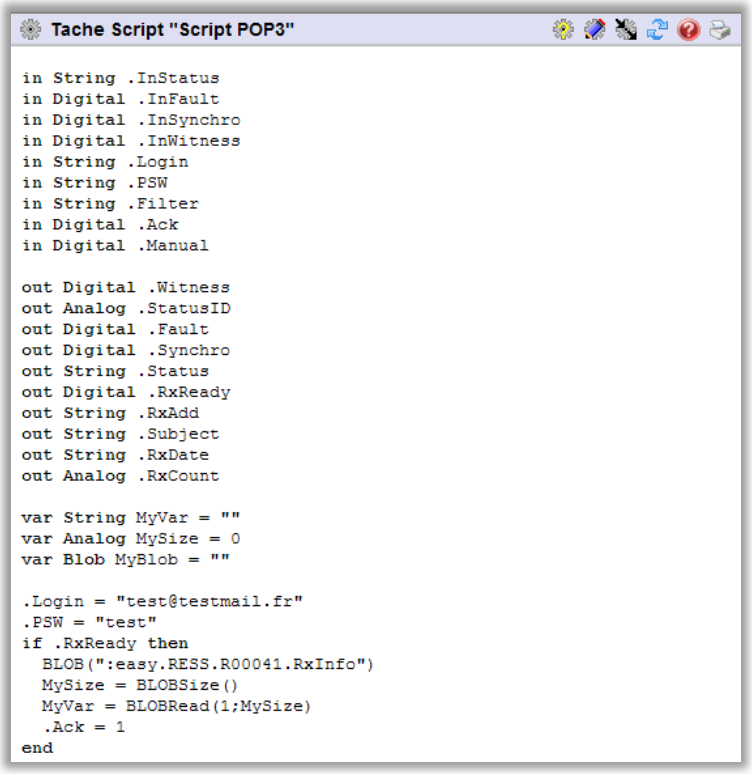
Login	Nom d'utilisateur de compte messagerie.
Psw	Nom de passe du compte messagerie.
Filter	Filtre sur l'expéditeur des adresses mails. Cette variable permet de renseigner les adresses e-mails des expéditeurs autorisés. Utiliser le caractère « pipe » (|) en séparateur de plusieurs adresses. **Ce paramètre est indispensable.**
Ack	Permet d'acquitter la prise en compte d'un mail, et ainsi passer au suivant. *Cette action a pour effet de supprimer le mail du serveur.*
Manual	Permet de forcer la réception des emails en-dehors de l'intervalle entre réception.

Variables de sortie

RxReady	Mail en attente de traitement.
RxAdd	Adresse e-mail de l'expéditeur.
Subject	Objet du mail.
RxDate	Date de réception.
RxCount	Nombre d'e-mails traités (après acquittement).

Exemple

Exemple de script permettant de récupérer les informations provenant d'un mail :



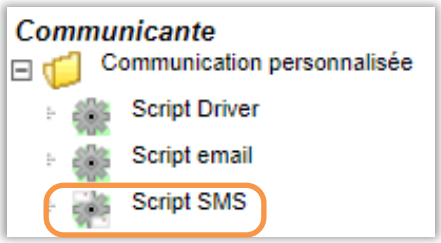
La ressource Script SMS

La ressource Script SMS permet de gérer la réception et émission de SMS.

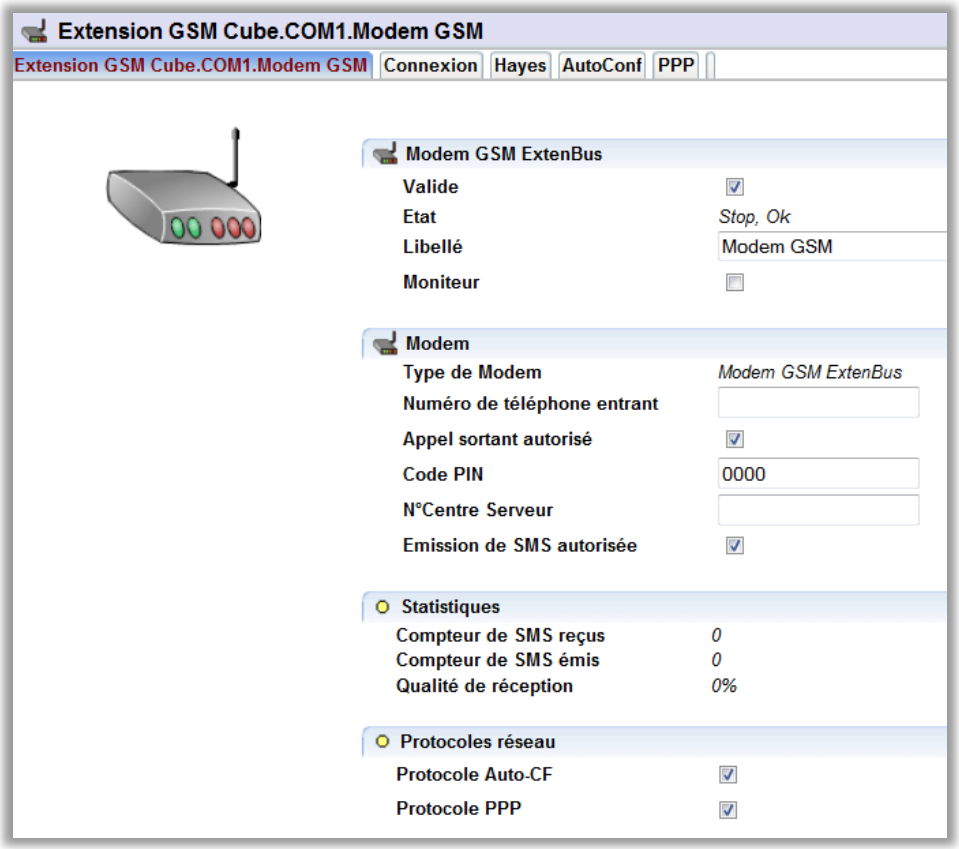
Cette ressource contient des variables d'entrée/Sortie pré-créées qui sont spécialisées dans le traitement de SMS.

Paramétrage de la ressource

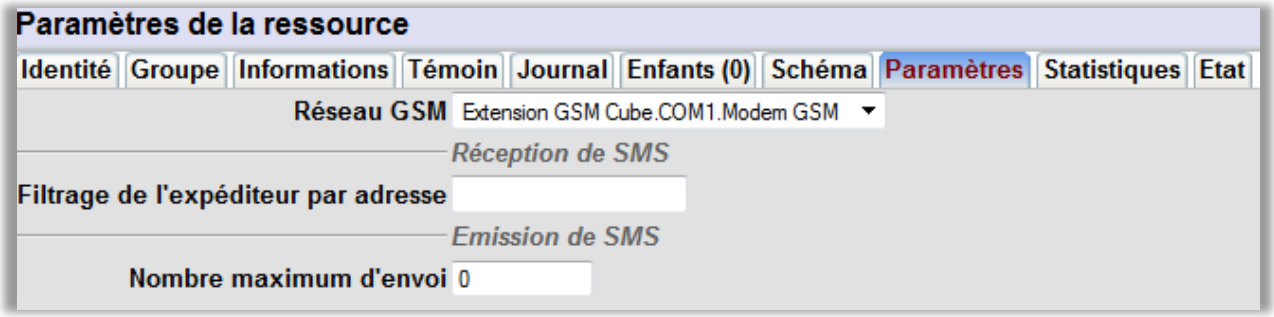
Etape 1 Ajouter une ressource Script SMS depuis le menu Paramétrage ► Ressource ► Ajouter une ressource ► Communication personnalisée ► Script SMS



Etape 2 Configurer le modem GSM en activant l'envoi de SMS si besoin depuis le menu Configuration ► Réseau ► [modem associé]
Exemple pour un GSM Cube :



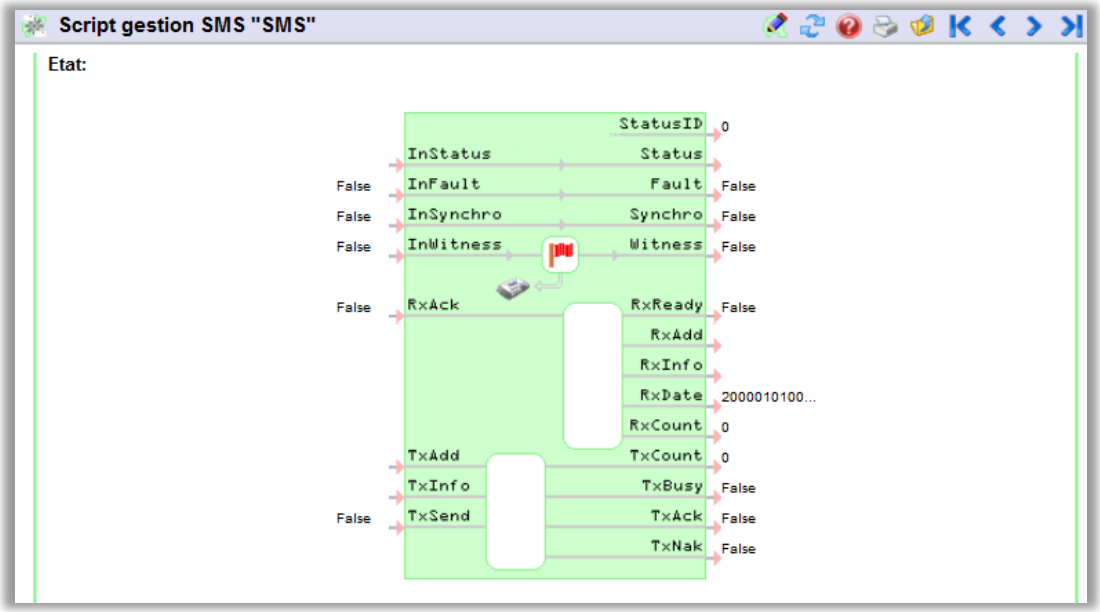
Etape 3 Lier le modem au Script SMS.



Il est possible d'ajouter un filtre de réception (n'acceptent les SMS provenant d'un seul numéro), et également de limiter le nombre d'envoi maximum de SMS émis par le Script.

Un seul **Script gestion SMS** ne peut être utilisé par modem GSM.

Liens de la ressource



Lors de la création de cette ressource, des variables d'entrées/sorties sont déjà présentes :

Variables d'entrée

RxAck	Permet d'acquitter la prise en compte d'un SMS, et ainsi passer au suivant. *Cette action a pour effet de supprimer le SMS.*
TxAdd	Numéro de téléphone vers lequel on transmet.
TxInfo	Contenu du message à transmettre.
TxSend	Permet d'envoyer le message contenue dans **TxInfo** au numéro contenu dans **TxAdd**.

Variables de sortie

RxReady	SMS en attente de traitement.
RxAdd	Numéro de l'expéditeur.
RxInfo	Message contenu dans le SMS reçu.
RxDate	Date de réception.
RxCount	Nombre de SMS traités (après acquittement).
TxCount	Nombre de SMS envoyé.
TxBusy	Actif durant l'envoi d'un SMS
TxAck	Passe actif si transmission OK
TxNak	Actif si erreur de transmission au serveur SMS

Exemple

Pour ce type de Script comme pour le tout Script de communication, il est conseillé de travailler en « étape ».

```
// ETAPE 3 : Attente d'acquittement
if (MyStep = 3) then
// Réception du SMS d'acquittement
if .RxReady then
if (.RxInfo = .Psw) then
MyAck = 1
.InStatus = "Evènement #"&&MyNewID&&" acquitté par : "&&.RxAdd
EventNew(.InStatus;0;".")
wait = 5
MyStep = 4
end
else
// Suppression du message reçu en cas de non-conformité (mots de passe incorrect)
.RxAck = 1
wait = 1
.RxAck = 0
end
end
```

Les routines (Annexe 1)

L'ULI permet d'abriter des routines écrite en langage de programmation « Scripts ».
Ces routines ont pour rôle d'ajouter des « Fonctions » manquante.

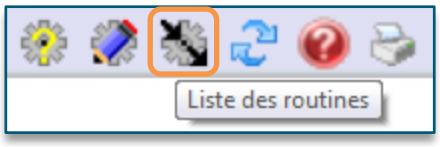
Les différences

Les différences avec une ressource Script :

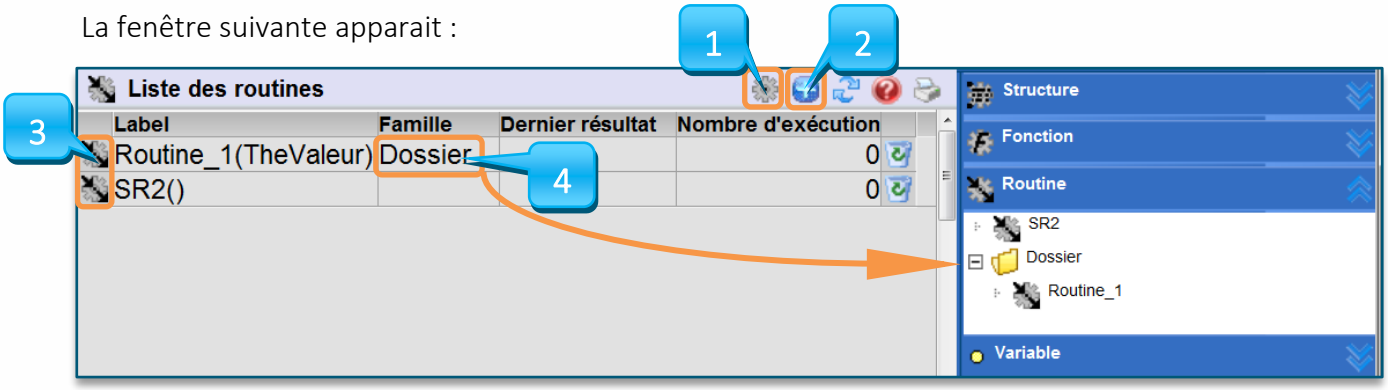
- On ne peut pas ajouter de liens d'entrée/sortie à une routine.
- L'exécution d'une routine est activée par un appel à partir d'un autre script.
- Une routine rend un résultat dans la variable « **result** ».
- On ne peut faire aucun calcul avec « **result** ».
- Le script est exécuté d'une manière synchrone mais avec une temporisation maximum de 5 secondes.

Création

Etape 1 Accéder à la liste des routines à travers une ressource Script en cliquant sur le bouton **Liste des routines**.



La fenêtre suivante apparait :



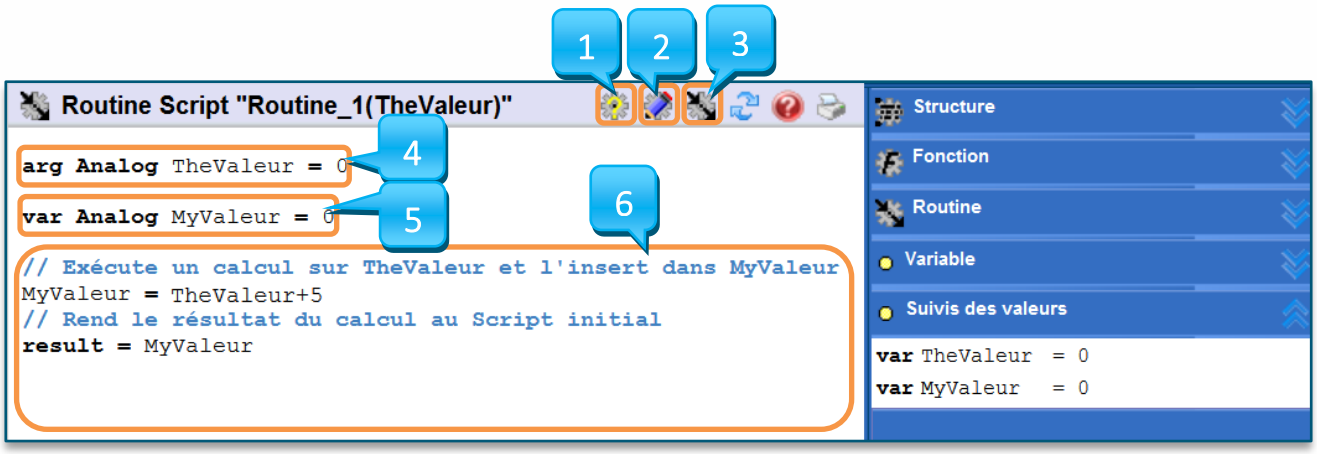
1 : Retour au Script d'origine
2 : Ajout d'une Routine
3 : Edition du Script de la Routine
4 : Dossier de rangement de la Routine

Une nouvelle Routine apparait sous le nom **SR** suivit du numéro de la routine avec les parenthèses vide.

Une fois la routine paramétrée, le nom de(s) la variable(s) argument est remonté entre les parenthèses.

Etape 2 Accéder à la routine

Une fois la routine créée, accéder à son Script en appuyant sur le bouton d'édition : 
La fenêtre suivante s'affiche :

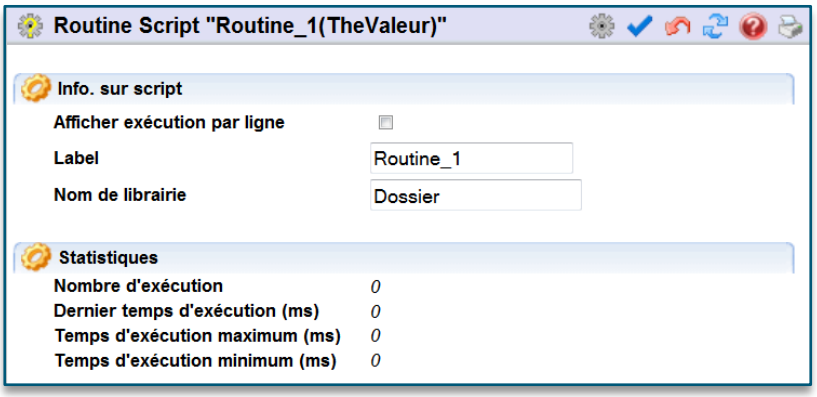


1 : Information est paramétrage de la Routine
2 : Edition du Script de la Routine
3 : Accéder à la lister des Routines
4 : Déclaration d'entrée(s) de la Routine
5 : Déclaration des variables internes
6 : Corps du Script

Une Routine peut posséder plusieurs **Entrées** *(nommées Augments)*, mais ne peut avoir qu'un seul résultat.

Etape 3 Paramétrer la routine

En cliquant sur le bouton **Info sur Script**, accéder au paramétrage de la Routine, la fenêtre suivante apparait :



Activer ou non l'affichage du nombre d'exécution par ligne (permet de trouver une éventuelle erreur et vérifier que le script s'exécute correctement).

Nommer la Routine *(ce nom sera utilisé pour appeler la Routine dans le Script)*

Il est également possible d'attribuer cette Routine à une **Librairie** *(qui sera le dossier dans lequel elle sera classée, permet de rassembler plusieurs Routine qui traitent d'un même sujet)*

Exemple

Voici un exemple montrant l'utilisation de la routine créée ci-dessus.
*Rappel, la routine créé prend la valeur **Argument**, et lui ajoute 5.*

Tache Script "Script"

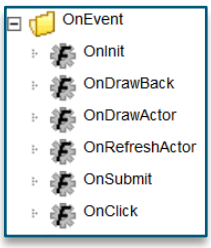
```
in String .InStatus
in Digital .InFault
in Digital .InSynchro
in Digital .InWitness

out Digital .Witness
out Analog .StatusID
out Digital .Fault
out Digital .Synchro
out String .Status

var Analog MyResultat = 10

if .InSynchro then
  MyResultat = Routine_1(5)
  .InSynchro = 0
end
```

Appelle de la **Routine_1** en lui indiquant une valeur d'argument : le chiffre **5**.

Le résultat dans la variable **MyResultat** est donc la valeur 10 *(5+5)*.

Scripts de synoptiques (Annexe 2)

Chaque synoptique abrite un script. Ce script est déclenché par les évènements suivants : « OnSubmit », « OnClick », « OnDrawBack », « OnDrawActor », « OnRefreshActor ».
Ces déclencheurs se retrouve dans le dossier « OnEvent » de la liste des fonctions.



Ces évènements mettent à jour les propriétés :

- ActorPath (Contenant le chemin de l'acteur).
- ActorLabel (Contenant le label de l'acteur).
- ActorValue (Contenant la valeur renvoyée par l'acteur)
- ActorID (Contient l'ID unique de l'acteur dans le synoptique)
- ActorX, ActorY (Contient la coordonnée X ou Y de la cellule cliquée pour un acteur tableau).
- SynURL (Si l'URL est renseignée, elle permet de rediriger le synoptique vers l'URL définie. Variable uniquement réinitialisée entre chaque appel)

Les déclanchements

Déclenchement par « OnSubmit »

Il intervient à chaque validation du formulaire (espace d'action dans l'interface) pour les acteurs suivants :

Type d'acteur	Représentation
Acteur ressource	Bouton CheckBox, Bouton poussoir, Icône.
Bouton	Avec l'option « Type d'action » à Valider.
Objet	Bouton CheckBox, Bouton poussoir, Icône.
Tableau	Avec au moins une colonne cliquable.

Ce déclenchement actualise les variables « ActorPath », « ActorLabel », « ActorValue », « ActorID », « ActorX », « ActorY », « SynURL ».

Déclenchement par « OnClick »

Il intervient à chaque click sur les acteurs suivants :

Type d'acteur	Représentation
Bouton	Avec l'option « Type d'action » à Hyper-Lien.
Liste de ressources	
Liste de zones	
Liste des évènements	
Menu	
Signalisation	

Ce déclenchement actualise les variables (« ActorPath », « ActorLabel », « SynURL »).

Déclenchement par « OnDrawBack »

Il intervient avant l'affichage du fond de la page synoptique.
Ce déclenchement actualise les variables « ActorValue », « SynURL ».

Déclenchement par « OnDrawActor »

Il intervient avant l'affichage de l'acteur.
Ce déclenchement actualise la variable « SynURL ».

Déclenchement par « OnRefreshActor »

Il intervient à chaque rafraichissement de l'acteur.
Ce déclenchement actualise la variables « SynURL ».

Les liens

Il est possible de transférer des informations d'un synoptique à un autre en ajoutant à la fin du lien « ?ArgScript=Valeur »

Cet argument sera récupéré sur le script récepteur par « .ActorValue »

Exemple, le lien suivant transmet les 3 arguments suivant :
"DEP_DEP_5", "RET_DEP_5", " Départ ECS".

URL : /easy/SYN/SYN_1/SYN_3?ArgScript=DEP_DEP_5|RET_DEP_5|Départ ECS

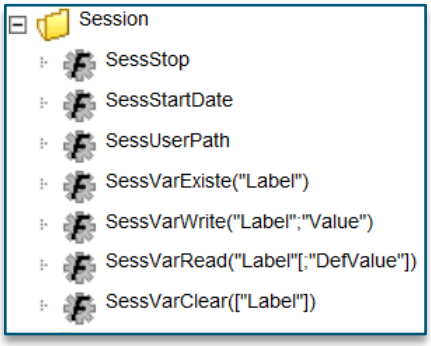
Une fois récupéré, il sera séparé par une fonction « StrField(String;FieldIndex[;SepChar]) ». Exemple : de la façon suivante : StrField(.ActorValue;1;"|") pour récupérer le premier argument.

Les variables de session

Elles permettent de travailler en fonction de la (les) personne(s) connectée(s) sur le serveur :

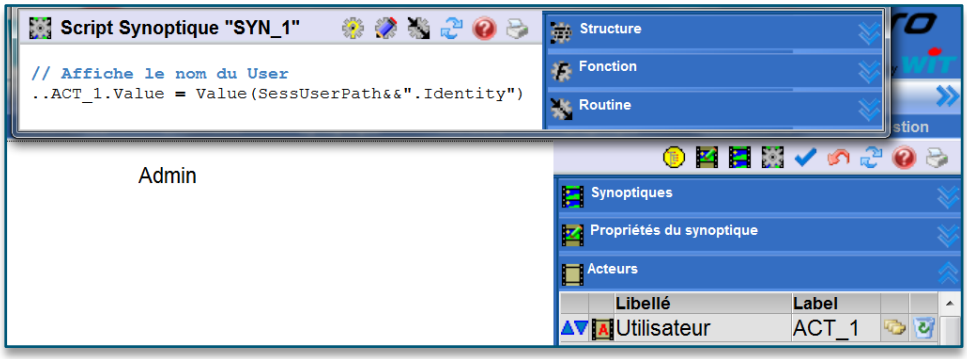
- SessStartDate : Cette fonction retourne la date de début d'ouverture de la session sous un format WitTime.
- SessStop : Cette fonction permet d'arrêter la session en cours.
- SessUserPath : Cette fonction permet de connaître le chemin, dans l'arborescence du produit, du User connecté (ex. :System.User.Admin).
- SessVarExiste : Cette fonction permet de tester l'existence d'une variable de session (ex. SessVarExiste(« Sess_Site »)).
- SessVarWrite : Cette fonction permet de créer et d'écrire dans une variable de session
- SessVarRead : Cette fonction permet de lire une variable préalablement créée.
- SessVarClear : Cette fonction permet de détruire une variable de session.

Ces variables se trouvent dans le dossier **Session** de la liste des fonctions :



Exemple

Voici un exemple de script synoptique qui permet d'afficher le nom de l'utilisateur connecté dans un acteur texte.



Pour tout renseignement complémentaire, notre support technique se tient à votre disposition par e-mail à hot-line@wit.fr ou par téléphone au +33 (0)4 93 19 37 30