



# Manuel d'utilisation Scripts



DTE037F - V1.1 - 07/14



Home II - 138.Avenue Léon Bérenger 06706 – Saint Laurent du Var Cedex  
Tel : 04 93 19 37 37 - Fax : 04 93 07 60 40 - Hot-line : 04 93 19 37 30  
Site : [www.wit.fr](http://www.wit.fr)

## SOMMAIRE

<b>Présentation</b> .....	<b>5</b>
<b>Généralités</b> .....	<b>5</b>
<b>Prérequis</b> .....	<b>5</b>
<b>Domaines d'application</b> .....	<b>5</b>
Les Ressources .....	5
Les Synoptiques.....	5
<b>Avertissement</b> .....	<b>6</b>
<b>Création de la ressource</b> .....	<b>7</b>
<b>Paramétrage</b> .....	<b>7</b>
<b>Edition</b> .....	<b>7</b>
<b>Définitions</b> .....	<b>8</b>
La déclaration des Variables .....	8
Interface .....	9
Mise au point du script .....	11
<b>Avant de commencer</b> .....	<b>12</b>
<b>La syntaxe</b> .....	<b>12</b>
<b>Les constantes</b> .....	<b>12</b>
<b>Les opérateurs et comparateurs</b> .....	<b>12</b>
<b>Chemin ou Path</b> .....	<b>13</b>
<b>Variable dynamique</b> .....	<b>13</b>
<b>Variables de Paramètres</b> .....	<b>14</b>
<b>Les structures</b> .....	<b>15</b>
<b>Conditionnelles</b> .....	<b>15</b>
<b>Spécifique</b> .....	<b>15</b>
<b>La liste des Fonctions commune e@sy / e@sy-Pilot</b> .....	<b>16</b>
<b>Affectation de valeurs</b> .....	<b>16</b>
<b>Les fonctions de traitement analogique</b> .....	<b>16</b>
<b>Les fonctions de traitement des Blobs</b> .....	<b>17</b>
<b>Les fonctions de traitement des chaînes</b> .....	<b>18</b>
<b>Les fonctions de traitement des dates</b> .....	<b>19</b>
<b>Les fonctions de conversion</b> .....	<b>20</b>
<b>Les fonctions de traitement des booléens</b> .....	<b>20</b>

Les fonctions diverses.....	20
Les fonctions de traitement des ensembles .....	21
Les fonctions logarithmiques.....	21
Sur événement.....	21
Les fonctions de traitement des évènements .....	22
Les fonctions de traitement des fichiers .....	23
Fonctions de traitement des tableaux .....	24
Ressource .....	24
Les fonctions de traitement des objets .....	25
Fonctions de traitement des traces .....	26
Fonction trigonométriques .....	26
Fonction WEB.....	27
Fonction WOD.....	27
<b>Spécifique E@sy .....</b>	<b>28</b>
Fonction FlashDisk .....	28
Les fonctions de traitement des objets .....	28
Les fonctions de traitement des chaînes .....	28
<b>Spécifique E@sy-pilot .....</b>	<b>29</b>
Les fonctions de traitement des fichiers.....	29
Fonctions de traitement des traces .....	29
Les fonctions de traitement SQL .....	30
<b>La ressource Script Driver .....</b>	<b>31</b>
Paramétrage de la ressource.....	31
Fonctions .....	32
<b>La ressource Script gestion POP3.....</b>	<b>33</b>
Paramétrage de la ressource.....	33
Liens de la ressource.....	34
Exemple.....	35
<b>La ressource Script gestion SMS .....</b>	<b>36</b>
Paramétrage de la ressource.....	36
Liens de la ressource .....	37
Exemple.....	38
<b>Les routines (Annexe 1).....</b>	<b>39</b>
Les différences .....	39

Création ..... 39

Exemple..... 41

**Scripts de synoptiques (Annexe 2)..... 42**

**Les déclenchements ..... 42**

    Déclenchement par « OnSubmit » ..... 42

    Déclenchement par « OnClick »..... 43

    Déclenchement par « OnDrawBack » ..... 43

    Déclenchement par « OnDrawActor » ..... 43

    Déclenchement par « OnRefreshActor » ..... 43

**Les liens ..... 43**

**Les variables de session ..... 44**

Exemple..... 44

## Présentation

### Généralités

L'e@sy intègre 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 un synoptique.

### 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 Scripts sont présents dans plusieurs fonctionnalités de l'e@sy.

#### 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 gestion SMS

La ressource « Script gestion SMS » permet à un une personne d'**interagir avec un e@sy par SMS<sup>1</sup>** : interrogation d'états, acquittement d'alarmes, ordre de commande, etc.



##### Script gestion POP3

La ressource « Script gestion POP3 » permet de gérer la **réception d'e-mails** depuis un serveur de messagerie POP.

#### Les Synoptiques



##### Script Synoptique

Le « Script Synoptique »

<sup>1</sup> Nécessite un e@sy intégrant un modem GSM (PLUG GSM) ou une Extension GSM Cube.

### Avertissement

---

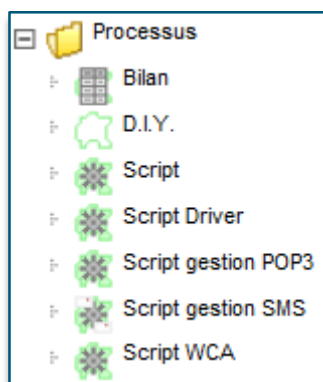
Une mauvaise utilisation des Scripts peut altérer le bon fonctionnement de l'e@sy : ralentissement de la navigation web et des automatismes par une saturation de la mémoire vive (RAM), blocage ou redémarrage dans le cas d'un Script bouclant indéfiniment sur lui-même, etc.

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

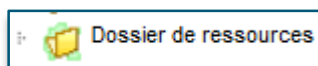
Pour connaître le programme et les dates de nos formations, vous pouvez nous contacter à [wit@wit.fr](mailto:wit@wit.fr).

## Création de la ressource

**Etape 1** Insérer la ressource Script correspondante depuis le menu **Paramétrage** ► **Ressource** ► **Ajouter une ressource** ► **Processus**



Dans un souci de clarté dans l'arborescence des ressources de l'e@sy, 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 votre ressource Script :

<b>Identité</b>	Groupe	Informations	Témoin	Journal	Enfants (0)	Schéma	Etat
Valide <input checked="" type="checkbox"/>							
Libellé Nom du Script							

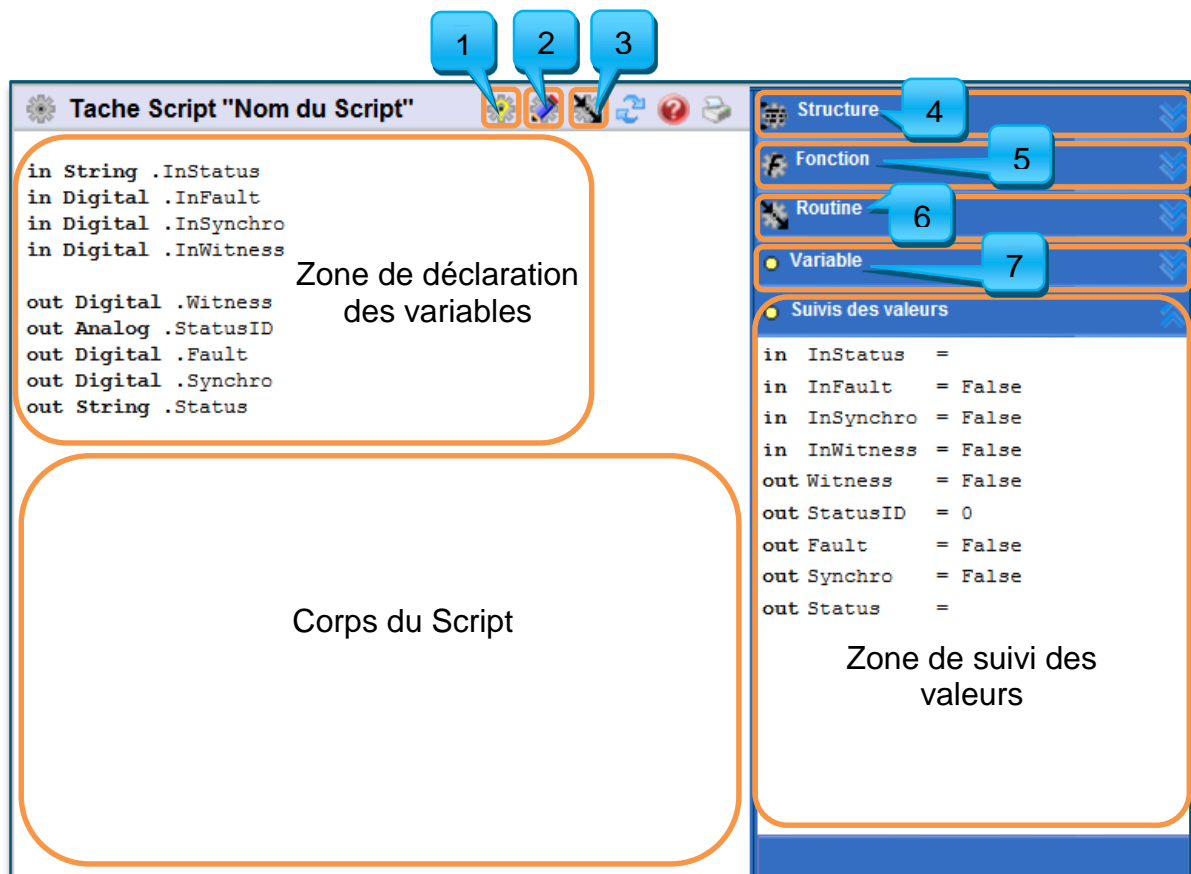
## Paramétrage

### 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 signaler 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 :

The screenshot shows a script editor window titled "Tache Script 'Nom du Script'". The main editor area contains the following code:

```

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 Digital MyEtat = False
var Analog MyNbExecution = 0
var String MyTexte = ""
var Blob MyPlan = ""
  
```

On the right side, there is a "Structure" panel with a "Suivis des valeurs" (Value Tracking) section. It lists the variables and their current values:

```

in InStatus =
in InFault = False
in InSynchro = False
in InWitness = False
out Witness = False
out StatusID = 0
out Fault = False
out Synchro = False
out Status =
var MyEtat = False
var MyNbExecution = 0
var MyTexte =
var MyPlan =
  
```

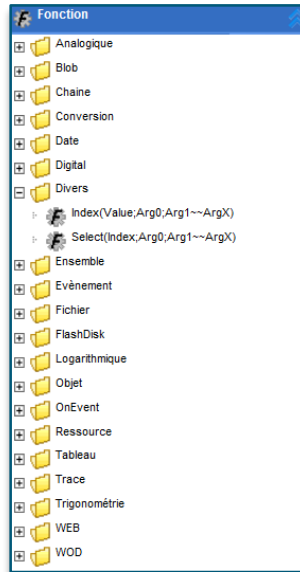
La zone de déclaration des variables apparaît en tête du corps du script. Les mêmes variables se retrouvent dans la zone de 'Suivis des valeurs', ce qui permet de suivre leur évolution au cours du déroulement du script.

## Interface

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



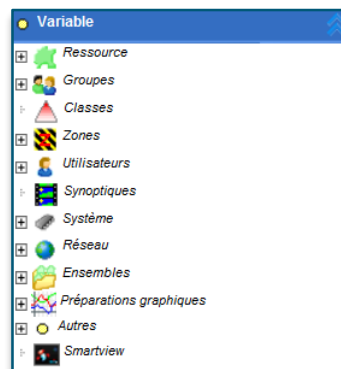
- Fonction : Dans cet onglet ce trouve toutes les fonctions disponible pour le traitement d'une valeur, les calculs etc...



- Routine : De base, cet onglet est vide, c'est ici que ce trouverons les routines que vous aurais créés, ces routine sont des petit bout de programme qui vienne compléter les actions qui ne sont pas réaliser par les fonctions déjà existantes.



- Variable : Ici vous trouverais tous les ressources qui sont présent dans l'e@sy



**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 :

```

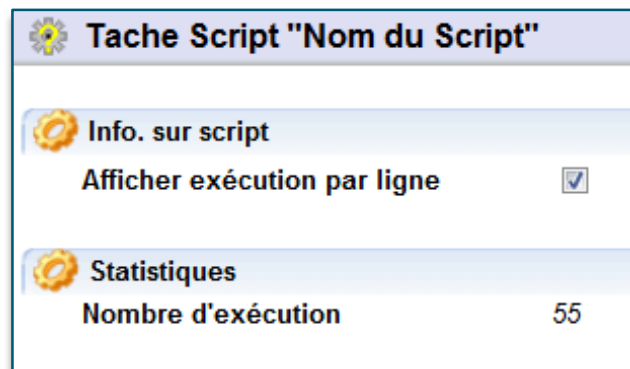
if .InStatus then
  // Affectation de zone
  MyIndex = 1
  while MyIndex<16 do
    @":easy.RESS.ExtenBUS.EXT001.DI"
    MyIndex = MyIndex+1
  end
  .InStatus = 0
end
    
```

## Mise au point du script

Pour afficher les informations sur le déroulement du script, il faut 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 :

```
000002456 if .InStatus=1 then
           // Active une entrée
000000000 :easy.RESS.R00003.R00005.InLink = 1
           // Attend 5 secondes
000000000 wait = 5
           // Remet l'entrée à 0
000000000 :easy.RESS.R00003.R00005.InLink = 0
000000000 .InStatus = 0
           end
```

Le script a eu la main 2456 fois, mais n'a jamais été exécuté aucune autre ligne que le 1<sup>er</sup> test.

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

```
000004548 if .InStatus=1 then
           // Active une entrée
000000001 :easy.RESS.R00003.R00005.InLink = 1
           // Attend 5 secondes
000000001 wait = 5
           // Remet l'entrée à 0
000000001 :easy.RESS.R00003.R00005.InLink = 0
000000001 .InStatus = 0
           end
```

Le Script a bien été exécuté qu'une seule fois.

## 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.

## 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)
- <> : Différent
- <= : Inférieur ou égale
- >= : Supérieur ou égale

Exemple : `if MyAnalog<=5 then`

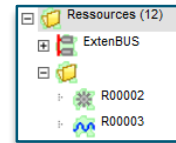
Pour concaténer 2 chaînes il faut utiliser les caractères &&

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 dans lesquels on entre.

Dans le cas où nos ressources sont classées selon l'ordre suivant :



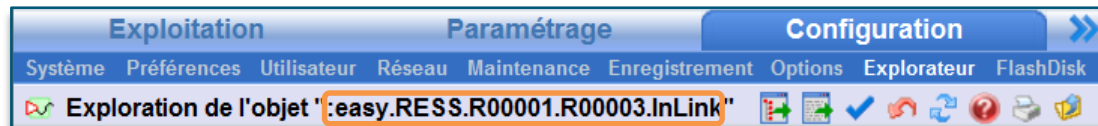
**Chemin relatif:** Son chemin est relatif par rapport au script

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 ressource, le chemin sera donc :  
`:easy.RESS.R00001.R00003.InLink = 54`

Pour trouver un chemin absolu complet, ou simplement le **LABEL** d'un dossier, il faut passer par **Configuration** ► **Explorateur** ► **easy** ► **RESS** ► (Cliquer ensuite sur le dossier qui vous intéresse puis les ressources enfants si désiré). Une fois arriver au lien voulu, vous trouverez juste en dessous du menu 2 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 relatifs seront toujours les mêmes dans le dossier dupliqué, le script dupliqué fonctionnera donc bien avec cette ressource associée, ce qui ne sera pas le cas dans le cas d'une adresse absolue.

## 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 15.0.0.0.

On peut faire :

```
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
```

La solution la plus approprié serait donc de faire 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
```

Si on veut affecter une valeur dont le chemin de la variable est reconstitué on utilise le @.  
@":easy.RESS.ExtenBUS.EXT001.DI"&&MyIndex&&".Zone" = ":System.Attribute.Zone.Z\_1"

Si on veut 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

Elles sont utilisées dans les routines. On les appelle « 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, on aura alors :

```
Routine Script "Modif_Ress(TheRacine)"
arg String TheRacine = ""
```

Le chemin est ici passé à l'argument TheRacine.

## Les structures

### Conditionnelles

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 <i>then</i> et le <i>end</i> 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 <i>then</i> et le <i>else</i> si la condition x est vrai ou les lignes de commandes entre le <i>else</i> et le <i>end</i> 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 <i>Repeat</i> et le <i>Until</i> 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 <i>Do</i> et le <i>End</i> 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.

## La liste des Fonctions commune e@sy / 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 a une entrée, sortie ou variable texte ou blob, celle-ci est insérée entre double guillemet.

*MyString = "Chaîne 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 valeur 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 default = 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 default = 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 <i>SubString</i> dans la chaîne <i>String</i>	Booléen	StrFind("T" ; "WIT")=True
StrPos(SubString;String)	Retourne la position de la sous-chaîne <i>SubString</i> dans la chaîne <i>String</i> . Retourne 0 si non trouvé.	Entier	StrPos("T" ; "WIT")=3
StrCopy(String;StartIndex[:;NbChar])	Extrait <i>NbChar</i> caractères à partir du <i>StartIndex</i> de la chaîne <i>String</i> .	Chaîne	StrCopy ("Formation" ;2 ;3)= "orm"
StrField(String;FieldIndex[:;SepChar])	Extrait le champ <i>FieldIndex</i> de la chaîne <i>String</i> dans laquelle le séparateur est <i>SepChar</i> .	Chaîne	StrField ("Toto;Tata;Titi";2 ;";") = "Tata"
StrFieldCount(String[:;SepChar])	Retourne le nombre de champs de la chaîne <i>String</i> dans laquelle le séparateur est <i>SepChar</i> .	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 chaîne <i>String</i>	Entier	
StrDecimal(Value[:;NbDecimal])	Force le nombre de décimale de <i>Value</i> à <i>NbDecimal</i> . (Bourrage avec des 0)	Chaîne	StrDecimal(4,3 ;4)= "4,3000"
StrDigit(Value[:;NbDigit])	Force le nombre de Digit de <i>Value</i> à <i>NbDigit</i> (Bourrage avec des 0)	Chaîne	StrDigit(23 ;4) = "0023"
StrVisible(String)	Supprime les caractères invisibles de la chaîne <i>String</i>	Chaîne	
StrLabel(String)	Modifie la chaîne <i>String</i> pour quelle puisse être utilisée en tant que Label d'un Objet	Chaîne	
StrPatchChar(String;Char;CharPatch)	Remplace le caractère <i>Char</i> de la chaîne <i>String</i> par le caractère <i>CharPatch</i> .	Chaîne	StrPatchChar ("TOTO";"O";"A") = "TATA"
StrDeleteChar(String[:;Left[:;Body[:;Right[:;Char]]])	Supprime le caractère <i>Char</i> de la chaîne <i>String</i> 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 <i>Value</i> en chaîne de caractère selon le <i>DataType</i>	Chaîne	StrHexaFromValue (10 ;#String)= "3130"
StrHexaToValue(String;DataType)	Convertit la chaîne <i>String</i> selon le <i>DataType</i>	Datatype	StrHexaToValue ("10";#long)=16
Char(NumChar)	Retourne le caractère Ascii de <i>NumChar</i>	Chaîne	Char(67)="C"
CharNum(Char)	Retourne la valeur du caractère Ascii <i>Char</i>	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 <i>WitTime</i>	Chaîne	Year(Clock)=7
Month(WitTime)	Retourne le mois du <i>WitTime</i>	Chaîne	
Day(WitTime)	Retourne le Jour du <i>WitTime</i>	Chaîne	
Hour(WitTime)	Retourne l'heure du <i>WitTime</i>	Chaîne	
Minute(WitTime)	Retourne les minutes du <i>WitTime</i>	Chaîne	
Second(WitTime)	Retourne les secondes du <i>WitTime</i>	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 <i>WitTime</i> au format JJ/MM/AAAA ou JJ/MM/AA si <i>short</i> est renseigné	Chaîne	Date(Clock ;1)= "31/10/07"
Time(WitTime)	Heure du <i>WitTime</i> au format HH :MM :SS	Chaîne	
DateTime(WitTime)	Date et Heure du <i>WitTime</i> au format JJ/MM/AAAA HH :MM :SS	Chaîne	
Clock	Nombre de secondes écoulées depuis le 1 <sup>er</sup> Janvier 2000	Entier	
ClockGMT	Nombre de secondes écoulées depuis le 1 <sup>er</sup> 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 <i>WitTime</i> en une chaîne au format AAAAMMJJHHMMSS	Chaîne	WitTimeToYMDHMS (Clock)= « 20071031105613 »
WitTimeOfYMDHMS(« YMDHMS »)	Convertit la chaîne AAAAMMJJHHMMSS en un <i>WitTime</i>	Entier	
WitTimeOfDate(« DD/MM/YYYY »)	Convertit la chaîne « DD/MM/YYYY » en un <i>WitTime</i>	Entier	
WitTimeOfTime(« HH :MM :SS »)	Convertit la chaîne « HH :MM :SS » en un <i>WitTime</i>	Entier	
WitTimeOfDateTime («DD/MM/YYYY HH:MM:SS»)	Convertit la chaîne « DD/MM/YYYY HH :MM :SS » en un <i>WitTime</i>	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 elle doit travailler.

Pour cela il faut 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 <i>Base</i> de <i>Value</i>	Réel
Log2(Value)	Retourne le Logarithme à Base 2 de <i>Value</i>	Réel
Log10(Value)	Retourne le Logarithme à Base 10 de <i>Value</i>	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 événements

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 événement	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 précédent	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>>^R Ress: ^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'évènement	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")	Teste si le fichier <i>FileName</i> existe
FileDate("FileName")	Retourne la date du fichier <i>FileName</i>
FileSize("FileName")	Retourne la taille du fichier <i>FileName</i>
FileCopy("FileNameSrc";"FileNameDest")	Copie le fichier <i>FileNameSrc</i> dans le fichier <i>FileNameDest</i>
FileAppend("FileNameSrc";"FileNameDest")	Copie le fichier <i>FileNameSrc</i> à la fin du fichier <i>FileNameDest</i>
FileClear("FileName")	Clare le fichier <i>FileName</i>
FileDelete("FileName")	Supprime le fichier <i>FileName</i>
FileRename("FileNameOld";"FileNameNew")	Renomme le fichier <i>FileName</i>
FileWrite("FileNameDest";"StringValue")	Ecrit la Chaîne <i>StringValue</i> dans le fichier <i>FileNameDest</i>
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 que vous voulez mettre.	Ecrit le <i>FileNameSrc</i> dans l' <i>ObjectSrc</i>
FileFromObject("ObjectSrc";"FileNameDest"[:;Append])	Ecrit l' <i>ObjectSrc</i> dans le <i>FileNameSrc</i>
DirectoryGet	Retourne le nom du dossier de travail
DirectorySet("DirectoryPath")	Force le dossier <i>DirectoryPath</i> 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 = 100 YMax e@syPilot = 10000 YMax e@sy = 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' <i>Object</i>	Chaîne	ParentLabel (":easy.RESS.R00003.R00006") =R00003
ParentPath(« Object »)	Chemin complet du Parent de l' <i>Object</i>	Chaîne	ParentPath (":easy.RESS.R00003.R00006") =:easy.RESS.R00003
ObjectLabel(« Object »)	Label de l' <i>Object</i>	Chaîne	ObjectLabel (":easy.RESS.R00003.R00006") =R00006
ObjectPath(« Object »)	Chemin complet de l' <i>Object</i>	Chaîne	ObjectPath (":easy.RESS.R00003.R00006") =:easy.RESS.R00006
ObjectURL(« Objet »)	Donne l'URL l' <i>Object</i>	Chaîne	ObjectURL(":easy.RESS.R00003.R00007") =/easy/RESS/R00003/R00007
ObjectCaption(« Objet »)	Donne le libellé de l' <i>Object</i>	Chaîne	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' <i>Object</i>	Chaîne	ObjectClass (":easy.RESS.R00003.R00006") =559
ObjectChange("Object")	Simule le changement de l' <i>Object</i> (non utilisé)		
ChildCreate("Object";"Label";[Class])	Permet de créer un objet de classe <i>Class</i> de Nom <i>Label</i> enfant de l' <i>Object</i> parent	Chaîne	
ChildCount("Object";Class)	Retourne le nombre d'enfant de classe <i>Class</i> enfants de l' <i>object</i> parent	Chaîne	
ChildLabel("Object";Index)	Retourne le label de l'enfant du parent <i>Object</i> dont le numéro d'ordre est <i>Index</i> . Le premier Objet enfant a 1 pour <i>Index</i> .	Chaîne	ChildLabel (":easy.RESS.R00003";3) =R00006 La ressource R00006 est le 3 <sup>e</sup> enfant de R00003.
ChildPath("Object";Index)	Retourne le chemin complet de l'enfant du parent <i>Object</i> dont le numéro d'ordre est <i>Index</i> . Le premier Objet enfant a 1 pour <i>Index</i> .	Chaîne	
ChildLabelList("Object";Class;"BlobDest")	Ecrit dans <i>BlobDest</i> la liste des labels des Enfants de classe <i>Class</i> dont le parent est <i>Object</i>	Chaîne	

ChildPathList (Object";Class;"BlobDest")	Ecrit dans <i>BlobDest</i> la liste des chemins des Enfants de classe <i>Class</i> dont le parent est <i>Object</i>	Chaîne	
DescendantPathList ("Object";Class;"BlobDest")	Ecrit dans <i>BlobDest</i> la liste des chemins des Descendants de classe <i>Class</i> dont le parent est <i>Object</i>	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' <i>Object</i>	Type de l'Objet	Value("easy."&&"RESS.R00003.R00006.Output") =19,6

## Fonctions de traitement des traces

Fonction	Signification
TraceFncExportBlob("TraceFnc";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.  TraceFncExportBlob(:easy.RESS.ExtenBUS.FNCT.F0001;Clock-3600;Clock)  Result=Nombre de Pas exporté

## Fonction trigonométriques

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

## Fonction WEB

Fonction	Signification	Type du Résultat	Exemple
WEBFileCreate("ObjectDir";"FileName")	Créer un fichier WEB enfant de ObjectDir,	Chaîne	Result=NodPath du fichier
WEBFolderCreate("ObjectDir";"FolderName")	Créer un dossier WEB enfant de ObjectDir,	Chaîne	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")	Recherche un fichier/folder WEB enfant de ObjectDir,	Chaîne	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 <i>ObjectSrc</i> avec le nom <i>NewName</i>
WODExist("ObjectSrc")	Teste si le WOD <i>ObjectSrc</i> existe
WODDelete("ObjectSrc")	Supprime le WOD <i>ObjectSrc</i>
WODDuplicate("ObjectSrc";"NewName")	Duplique le WOD <i>ObjectSrc</i> avec le nom <i>NewName</i>
WODPublic("ObjectSrc";#True #False[;#True])	Rend Public le WOD <i>ObjectSrc</i> Le 3° Argument indique si on veut répercuter l'affection aux enfants de <i>ObjectSrc</i>
WODCopyToBLOB("ObjectSrc";"BlobDest")	Copy le WOD <i>ObjectSrc</i> dans le BLOB <i>BLOBDest</i>
WODCopyValueToBLOB("ObjectSrc";"BlobDest")	Copy la valeur du WOD <i>ObjectSrc</i> dans le BLOB <i>BLOBDest</i>
WODPasteFromBLOB("BlobSrc";"ObjectDest")	Colle le BLOB <i>BLOBSrc</i> dans l'objet <i>ObjectDest</i>
WODAppendFromBLOB("BlobSrc";"ObjectDest";"Name")	Ajoute le BLOB <i>BLOBSrc</i> dans l'objet <i>ObjectDest</i> . Le nom de l'objet crée est <i>Name</i> .
WODCopyToFile("ObjectSrc";"FileNameDest")	Copy le WOD <i>ObjectSrc</i> dans le fichier <i>FileNameDest</i>
WODCopyValueToFile("ObjectSrc";"FileNameDest")	Copy la valeur du WOD <i>ObjectSrc</i> dans le fichier <i>FileNameDest</i>
WODPasteFromFile("FileNameSrc";"ObjectDest")	Colle le Fichier <i>FileNameDest</i> dans l'objet <i>ObjectDest</i>
WODAppendFromFile("FileNameSrc";"ObjectDest";"Name")	Ajoute le Fichier <i>FileNameDest</i> dans l'objet <i>ObjectDest</i> . Le nom de l'objet crée est <i>Name</i> .

## Spécifique e@sy

## Fonction FlashDisk

Fonction	Signification	Type du Résultat
FlashReady	Actif si la flash est prête à exécuter une action	Booléen
FlashFree	Donne la taille disponible de la RAM en octet	Entier
FlashFileSize("FileName")	Donne la taille de <i>FileName</i> en octet	Entier
FlashFileToBLOB("FileName")	Ecrit dans le Blob en cours le contenu du fichier <i>FileName</i>	Booléen
FlashFileFromBLOB("FileName")	Ecrit dans <i>FileName</i> le contenu du BLOB en cours	Booléen
FlashFileKill("FileName")	Efface <i>FileName</i>	Booléen
FlashFileExist("FileName")	Rend Vrais si le fichier <i>FileName</i> existe.	Booléen
FlashFileRename("FileNameOld"; "FileNameNew")	Renomme le fichier <i>FileNameOld</i> avec le nom <i>FileNameNew</i>	Booléen
FlashFileByIndex(Index)	Donne le nom du fichier numéroté <i>Index</i>	String

## Les fonctions de traitement des objets

Fonction	Signification	Type du Résultat	Exemple
ObjectIDByPath("Object")	Fournit le numéro ID de <i>Object</i>	Entier	ObjectIDByPath(":easy.RESS.R00002") = 1743
ObjectPathByID("ID")	Fournit le chemin correspondant à <i>ID</i>	Chaîne	ObjectPathByID("1776") = ":easy.RESS.R00002.Output"

## Les fonctions de traitement des chaînes

Fonction	Signification	Type du Résultat	Exemple
StrToSQLStr(String)	Formate la chaîne <i>String</i> de caractères au format chaîne SQL (Ajoute ')	Chaîne	StrToSQLStr("Chaîne d'initialisation") = "'Chaîne 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 <i>DirectoryPath</i> comme dossier de travail

## Fonctions de traitement des traces

Fonction	Signification
TraceBoolean("TraceName";Value[;WitTime])	Écrit dans la Trace <i>TraceName</i> la <i>Value</i> Booléenne avec la Date <i>WitTime</i>
TraceReal("TraceName";Value[;WitTime])	Écrit dans la Trace <i>TraceName</i> la <i>Value</i> Entière avec la Date <i>WitTime</i>
TraceString("TraceName";Value[;WitTime])	Écrit dans la Trace <i>TraceName</i> la <i>Value</i> Chaîne avec la Date <i>WitTime</i>
TraceFirstDate("TraceName")	Retourne en <i>WitTime</i> la date du premier échantillon de la Trace <i>TraceName</i>
TraceNextDate("TraceName" ;WitTime)	Retourne en <i>WitTime</i> la date de l'échantillon suivant l'échantillon correspondant au paramètre <i>WitTime</i> de la Trace <i>TraceName</i>
TraceLastDate("TraceName")	Retourne en <i>WitTime</i> la date du dernier échantillon de la Trace <i>TraceName</i>
TraceDate("TraceName";WitTime)	Retourne en <i>WitTime</i> la date de l'échantillon la plus proche de <i>WitTime</i> de la Trace <i>TraceName</i>
TraceValue("TraceName";WitTime ;Value)	Permet de changer la valeur d'un pas déjà existant et ayant comme date <i>WitTime</i> de la Trace <i>TraceName</i>
TraceValueChange("TraceName";WitTime)	Retourne la valeur de l'échantillon le plus proche de <i>WitTime</i> de la Trace <i>TraceName</i>
TraceExport("TraceNameSrc";"FileNameDest"[;WitTimeStart[;WitTimeEnd[;Info[;Append[;EndChar]]])	Exporte la Trace <i>TraceNameSrc</i> dans le fichier <i>FileNameDest</i>
TraceFullExport("TraceNameSrc";"FileNameDest"[;WitTimeStart[;WitTimeEnd[;Info[;Append[;EndChar]]])	Exporte la Trace <i>TraceNameSrc</i> dans le fichier <i>FileNameDest</i> . Réalise une extrapolation sur les échantillons non présents dans le fichier.
TraceBooleanImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Booléen de la Trace <i>FileNameSrc</i> dans le fichier <i>TraceNameDest</i>
TraceRealImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Réel de la Trace <i>FileNameSrc</i> dans le fichier <i>TraceNameDest</i>
TraceStringImport("FileNameSrc";"TraceNameDest"[;EndChar])	Réalise l'import Chaîne de la Trace <i>FileNameSrc</i> dans le fichier <i>TraceNameDest</i>

TraceAppend("TraceNameSrc";"TraceNameDest" [;WitTimeStart[;WitTimeEnd]])	Ajoute les échantillons du fichier <i>TraceNameSrc</i> à la fin du fichier <i>TraceNameDest</i>
TracePurge("TraceNameSrc"[;WitTimeStart[;WitTimeEnd]])	Réalise la purge de la Trace <i>TraceNameSrc</i>
TraceGetInfo("TraceName";[WitTimeStart[;WitTimeEnd]])	Lit les informations du fichier Trace <i>TraceName</i>  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 chaîne	Chaîne	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 remplit 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)	Chaîne	ChildCreate("","BlobSQL";40) BLOB(".BlobSQL") BLOBWrite("Select * from JRNL") result = SQLSelectBLOBToStr ()
SQLSelectBLOBtoTABLE([TimeOut])	Exécute un script SQL Select contenu dans un BLOB et remplit 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 chaîne	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	Chaîne	Result= « Nom d'objet 'LISTE_SITES' non valide »

## La ressource Script Driver

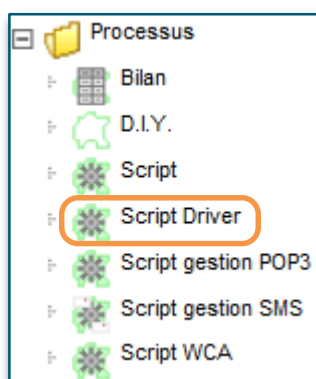
L'e@sy permet de réaliser son propre protocole de communication.

Pour cela, il suffit 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.

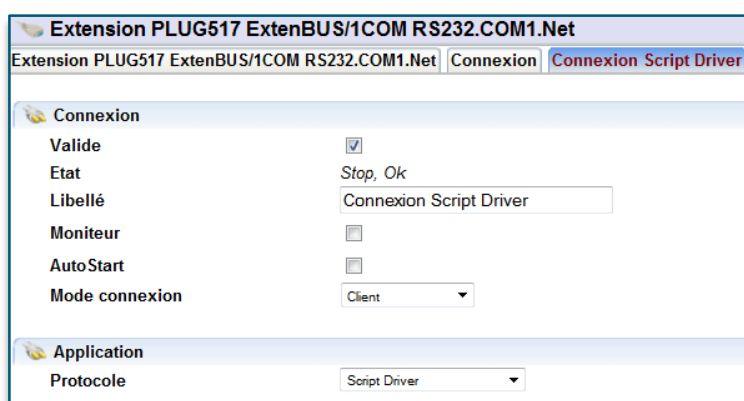
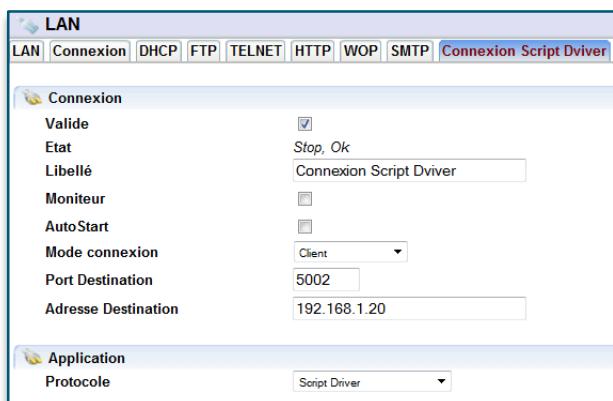
## Paramétrage de la ressource

**Etape 1** Ajouter une ressource **Script Driver** depuis le menu **Paramétrage** ► **Ressource** ► **Ajouter une ressource** ► **Processus** ► **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 en fonction du fonctionnement désiré. (*Auto Start non cocher dans le cas où la communication est a un moment précis, cocher si l'on attend une question sans savoir à quel moment précisément, dans le cas où l'on est esclave par exemple*)
- Le **Mode connexion** est en Client si on est à l'initiative de la communication (*Maitre*), est en serveur si l'on attend des informations (*Esclave*).
- Dans le cas d'une communication IP, définir le **Port destination**, et si l'on est maitre, l'**Adresse Destination** également.





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

**Paramètres de la ressource**

Identité | Groupe | Informations | Témoin | Journal | Enfants (0) | Schéma | **Paramètres** | Etat

Communication

Connexion Script LAN.Connexion Script Dviver

Taille du buffer réception (Octet) 1000

Taille du buffer émission (Octet) 200

Type de filtrage réception --

Texte Ascii avec CR et TAB

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

## Fonctions

Il y a dans cette ressource le dossier de ressource supplémentaire **Communication**.

Dans ce dossier ce trouve les fonctions suivantes :

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	Chaîne	Result=1° carcatère
RxString	Extrait une chaîne <= 255	Chaîne	Chaîne <= 255
RxLine	Extrait une chaîne <= 255 ayant comme délimiteur CR	Chaîne	Type de filtrage = CR Chaîne sans le CR Sans Type de filtrage Chaîne <= 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))



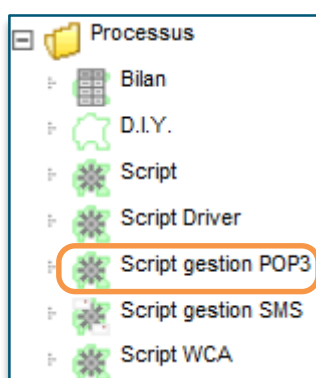
## La ressource Script gestion POP3

La ressource **Script gestion POP3** permet de récupérer les mails contenus dans un serveur de messagerie POP3 et de gérer leurs contenus. Cette ressource contient également le dossier de fonction **Communication**.

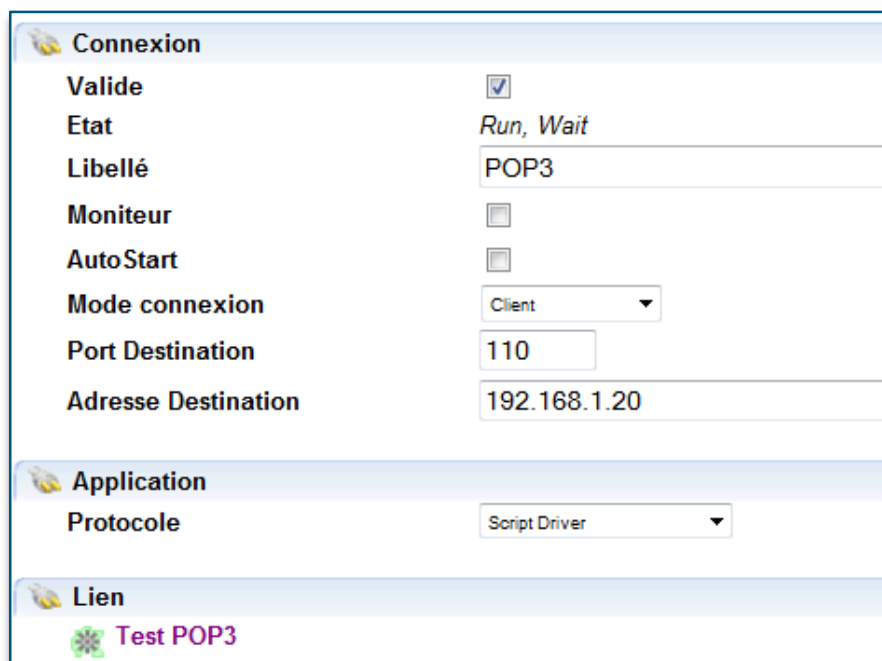
Le protocole **POP** (*Post Office Protocol*) permet comme son nom l'indique d'aller récupérer son courrier sur un serveur distant (le serveur POP).

### Paramétrage de la ressource

**Etape 1** Ajouter une ressource **Script gestion PO3** depuis le menu **Paramétrage ► Ressource ► Ajouter une ressource ► Processus ► Script gestion POP3**



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

A screenshot of a configuration dialog box titled 'Connexion'. It has three sections: 'Connexion', 'Application', and 'Lien'.  
- In the 'Connexion' section: 'Valide' is checked; 'Etat' is 'Run, Wait'; 'Libellé' is 'POP3'; 'Moniteur' is unchecked; 'AutoStart' is unchecked; 'Mode connexion' is 'Client'; 'Port Destination' is '110'; 'Adresse Destination' is '192.168.1.20'.  
- In the 'Application' section: 'Protocole' is 'Script Driver'.  
- In the 'Lien' section: there is a link icon and the text 'Test POP3'.

**Mode de connexion** : Client.

**Port de destination** : Le port par défaut est 110.

**Adresse de destination** : Adresse à laquelle se trouve le serveur POP3 sur le réseau.

**Protocole** : Choisir **Script Driver**

**Etape 3** Lier la connexion au **Script gestion POP3** et définir la taille des buffers de réception et d'émission.

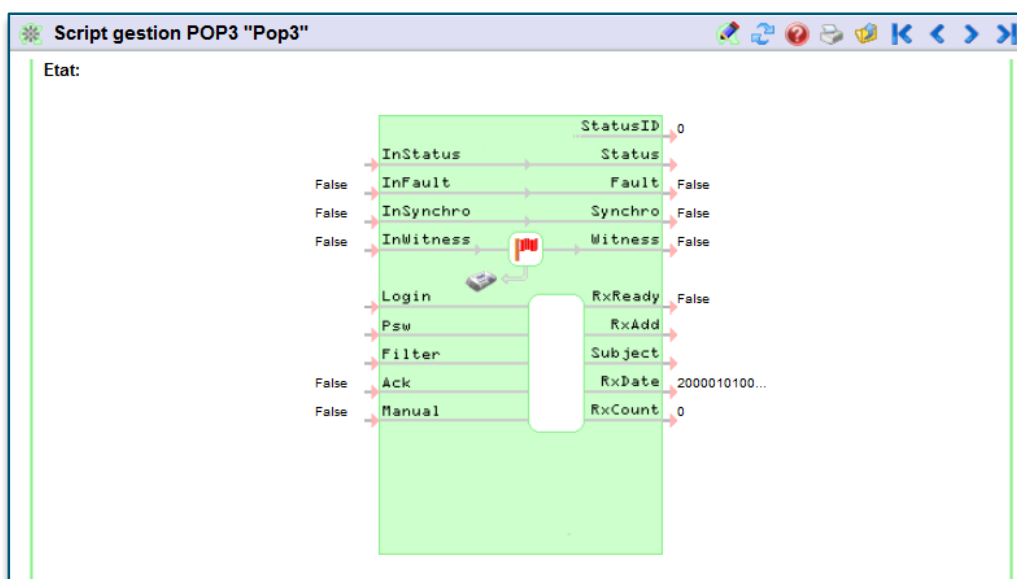
- Connexion Script :** Permet de lier la ressource à la connexion POP3.
- Taille du buffer de réception :** Valeur par défaut 1000 Octets. *Non utilisé*
- Taille du buffer d'émission :** Valeur par défaut 200 Octets. *Non utilisé*
- Type de filtrage réception :** Sans filtrage ou bien avec « Cr+LF ».
- 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

La ressource **Script gestion POP3** apparaît de la façon suivante.



Lors de la création de cette ressource, des variables d'entrées/sorties sont déjà présentes, celle-ci sont décrite ci-dessous.

## Variables d'entrée

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

## Variables de sortie

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

## Exemple

Voici un exemple de script permettant de récupérer les informations provenant d'un mail.

```

Tache Script "Script POP3"

in String .InStatus
in Digital .InFault
in Digital .InSynchro
in Digital .InWitness
in String .Login
in String .PSW
in String .Filter
in Digital .Ack
in Digital .Manual

out Digital .Witness
out Analog .StatusID
out Digital .Fault
out Digital .Synchro
out String .Status
out Digital .RxReady
out String .RxAdd
out String .Subject
out String .RxDate
out Analog .RxCount

var String MyVar = ""
var Analog MySize = 0
var Blob MyBlob = ""

.Login = "test@testmail.fr"
.PSW = "test"
if .RxReady then
  BLOB(":easy.RESS.R00041.RxInfo")
  MySize = BLOBSize()
  MyVar = BLOBRead(1;MySize)
  .Ack = 1
end

```

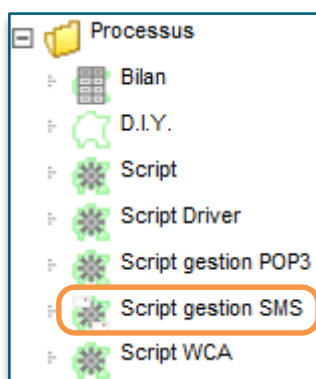
## La ressource Script gestion SMS

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

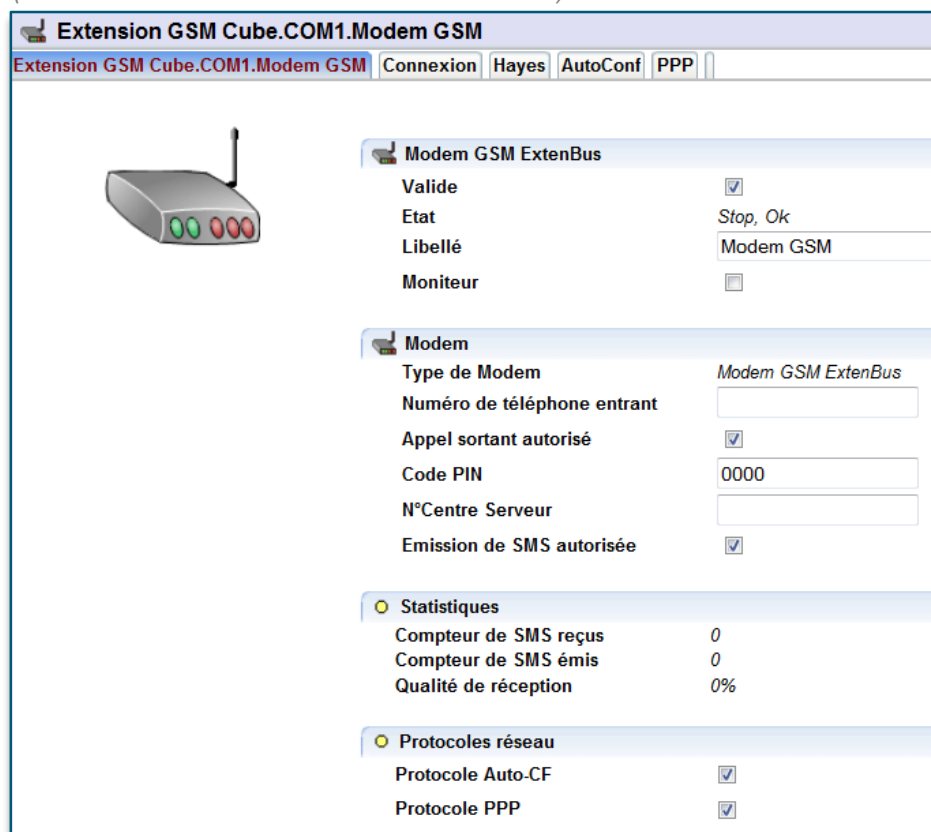
Cette ressource contient des variables d'Entrée/Sortie pré-crée qui sont spécialisées dans le traitement de SMS.

### Paramétrage de la ressource

**Etape 1** Ajouter une ressource **Script gestion SMS** depuis le menu **Paramétrage ► Ressource ► Ajouter une ressource ► Processus ► Script gestion SMS**



**Etape 2** Configurer le modem GSM en activant l'envoi de SMS si besoin depuis le menu **Configuration ► Réseau ► WAN** (dans le cas d'un modem GSM interne, PLUG601) ou **Extension GSM Cube** (dans le cas d'utilisation d'une extension GSM)



**Etape 3** Lier le modem au **Script gestion SMS**.

Paramètres de la ressource									
Identité	Groupe	Informations	Témoin	Journal	Enfants (0)	Schéma	Paramètres	Statistiques	Etat
Réseau GSM		Extension GSM Cube.COM1.Modem GSM							
<i>Réception de SMS</i>									
Filtrage de l'expéditeur par adresse		<input type="text"/>							
<i>Emission de SMS</i>									
Nombre maximum d'envoi		<input type="text" value="0"/>							

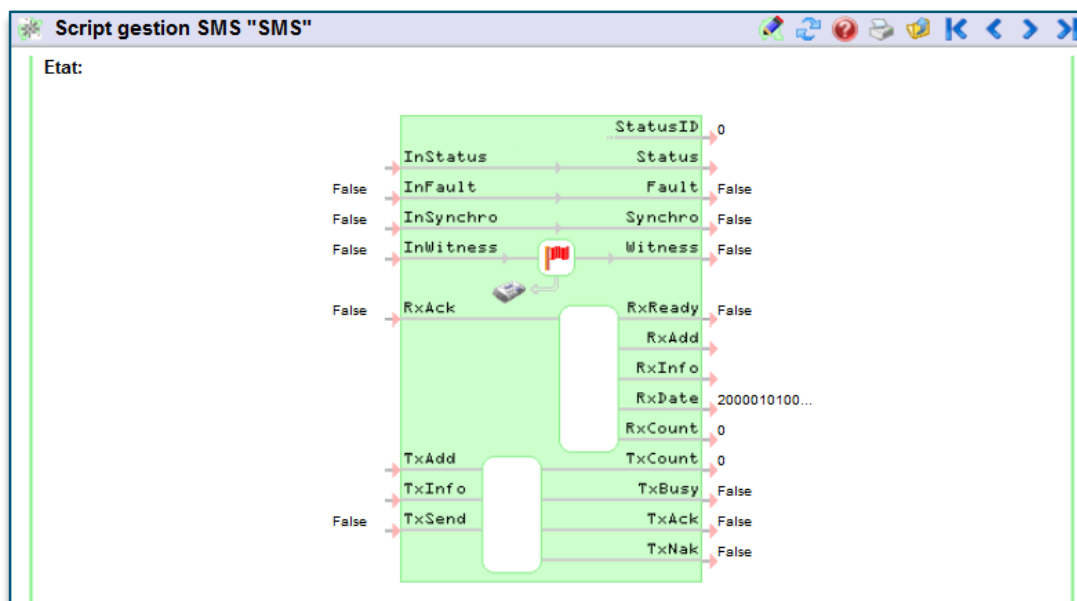
Il est possible d'ajouter un filtre de réception, n'acceptent les SMS provenant d'un seul numéro, et également de limiter le nombre d'envoi maximum de SMS émis par le Script



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

## Liens de la ressource

La ressource **Script gestion SMS** apparait de la façon suivante.



Lors de la création de cette ressource, des variables d'entrées/sorties sont déjà présentes, celle-ci sont décrite ci-dessous.

## Variables d'entrée

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

## Variables de sortie

<b>RxReady</b>	SMS en attente de traitement.
<b>RxAdd</b>	Numéro de l'expéditeur.
<b>RxInfo</b>	Message contenu dans le SMS reçu.
<b>RxDate</b>	Date de réception.
<b>RxCount</b>	Nombre de SMS traités (après acquittement).
<b>TxCount</b>	Nombre de SMS envoyé.
<b>TxBusy</b>	Actif durant l'envoi d'un SMS
<b>TxAck</b>	Passé actif si transmission OK
<b>TxNak</b>	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'acquiescement
if (MyStep = 3) then
// Réception du SMS d'acquiescement
if .RxReady then
if (.RxInfo = .Psw) then
MyAck = 1
.InStatus = "Evènement #"&&MyNewID&&" acquiescé 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'e@sy permet d'abriter des routines écrites en langage de programmation « Scripts ». Ces routines ont pour rôle d'ajouter des « Fonctions » manquantes à l'e@sy.

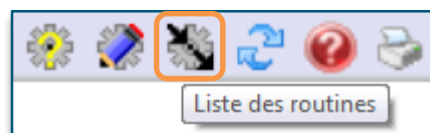
### Les différences

Les différences avec une ressource Script :

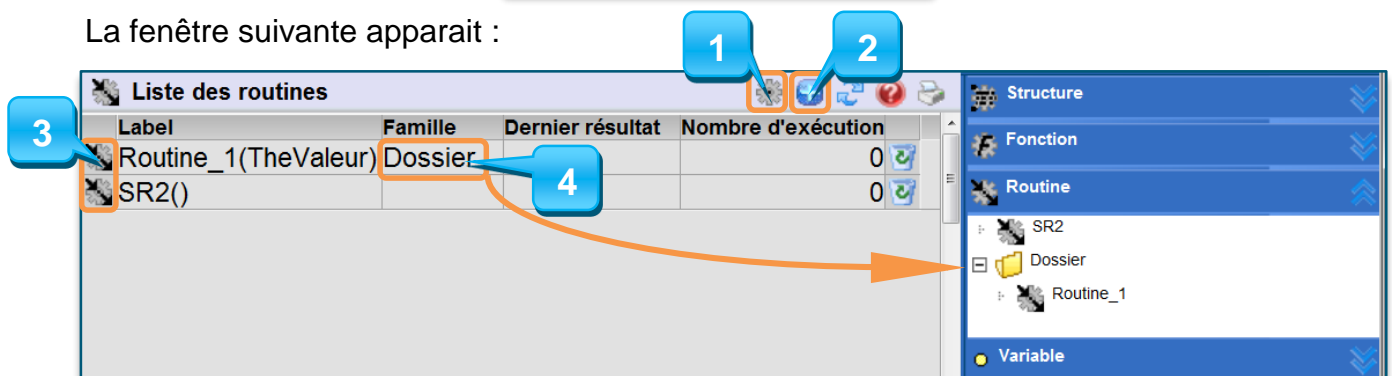
- On ne peut pas ajouter de liens d'entrée/sortie à une routine.
- L'exécution d'une routine est activée par un appel à partir d'un autre script.
- Une routine rend un résultat dans la variable « **result** ».
- On ne peut faire aucun calcul avec « **result** ».
- Le script est exécuté d'une manière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

### 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 apparaît :



- 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 apparaît sous le nom **SR** suivi du numéro de la routine avec les parenthèses vides.

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 créé, nous accédons a sont Script en appuyant sur le bouton d'édition :



La fenêtre suivante s'affiche :

La fenêtre de l'éditeur de routine "Routine\_1(TheValeur)" est affichée. Elle contient un code de script et une structure de routine. Les annotations numérotées indiquent les éléments suivants :

- 1 : Bouton d'information (roue dentée)
- 2 : Bouton d'édition (ciseaux)
- 3 : Bouton de liste des routines (carré avec flèche)
- 4 : Déclaration d'entrée `arg Analog TheValeur = 0`
- 5 : Déclaration de variable interne `var Analog MyValeur = 0`
- 6 : Corps du script (commentaires et ligne de code)

```

// Exécute un calcul sur TheValeur et l'insert dans MyValeur
MyValeur = TheValeur+5
// Rend le résultat du calcul au Script initial
result = MyValeur
  
```

La structure de routine à droite est :

- Structure
- Fonction
- Routine
- Variable
- Suivis des valeurs

Les variables suivies des valeurs sont :

```

var TheValeur = 0
var MyValeur = 0
  
```

- 1 : Information est paramétrage de la Routine
- 2 : Edition du Script de la Routine
- 3 : Accéder a 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 2** Paramétrer la routine

En cliquant sur le bouton **Info sur Script**, vous accéderez au paramétrage de la Routine, la fenêtre suivante apparaîtra :

La fenêtre de paramétrage de la routine "Routine\_1(TheValeur)" est affichée. Elle contient deux sections :

- Info. sur script**
  - Afficher exécution par ligne :
  - Label : Routine\_1
  - Nom de librairie : Dossier
- Statistiques**
  - Nombre d'exécution : 0
  - Dernier temps d'exécution (ms) : 0
  - Temps d'exécution maximum (ms) : 0
  - Temps d'exécution minimum (ms) : 0



Nous avons sur cette fenêtre la possibilité d'activer ou non l'affichage du nombre d'exécution par ligne, ce qui peut permettre de trouver une éventuelle erreur et vérifier que le script exécute bien est seulement les lignes qu'il doit en fonctionne des variable d'entrées.

Nous pouvons nommer la Routine (*ce nom sera utilisé pour appeler la Routine dans le Script*)

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

Il y a ensuite, dans la partie basse de cette page, les statistiques d'exécution de cette Routine.

## Exemple

Voici un exemple montrant l'utilisation de la routine créé 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
```

Nous voyons ci-dessus que nous appelons la Routine **Routine\_1** en lui mettent comme valeur d'argument, le chiffre **5**.

Nous avons donc bien comme résultat dans la variable MyResultat la valeur 10 ( $5+5$ ).

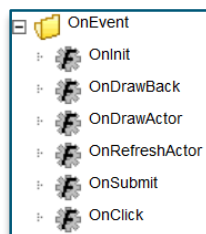
## Scripts de synoptiques (Annexe 2)

Chaque synoptique abrite un script.

Ce script est déclenché par les événements suivants :

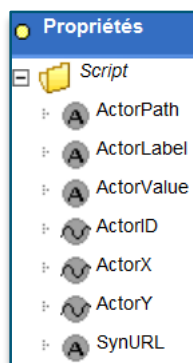
« OnSubmit », « OnClick », « OnDrawBack », « OnDrawActor », « OnRefreshActor ».

Ces déclencheurs se retrouvent dans le dossier « OnEvent » de la liste des fonctions.



Ces événements 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 déclenchements

### 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 a un autre en ajoutent à la fin du lien « ?ArgScript=Valeur »

Ceci 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

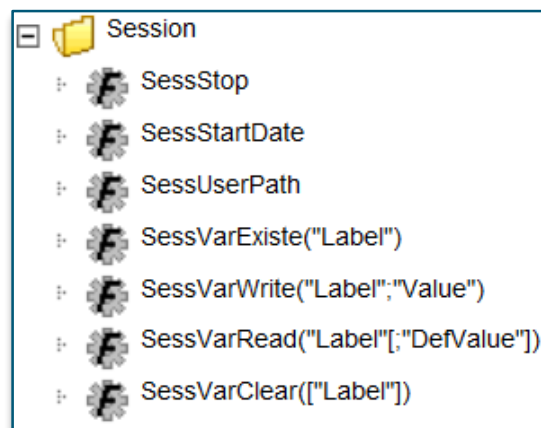
Qui une fois récupéré, il suffira de les séparer par une fonction « StrField(String;FieldIndex[;SepChar] ) ».  
Exemple : de la façon suivante : StrField(.ActorValue;1;"|") pour récupérer le premier argument.

## 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.

