HP Connect-It

Version du logiciel : 3.90

Référence de programmation

Date de publication de la documentation : 15 May 2008
Date de publication du logiciel : May 2008

Avis juridiques

Table des matières

I Introduction

1 Bases essentielles de programmation

Ce chapitre présente les composants fondamentaux du langage Basic disponible sous Connect-It. Si vous possédez des bases de programmation et que vous avez déjà pratiqué d'autres langages, la très grande majorité de ce chapitre vous sera familière. Nous vous invitons néanmoins à parcourir rapidement les différentes sections proposées, certaines fonctionnalités classiques étant volontairement limitées ou absentes du Basic de Connect-It.

Introduction aux variables

Les variables sont utilisées pour stocker des données au cours de l'exécution d'un programme. Elles sont identifiées par :

- leur nom, utilisé pour référencer la valeur contenue par la variable.
- leur type, qui détermine quelles données peuvent être stockées dans la variable.

On distingue en général deux types de variables :

- Les tableaux,
- Les variables scalaires qui regroupent toutes les variables qui ne sont pas des tableaux.

Déclarer une variable

Toute variable doit être explicitement déclarée avant d'être utilisée. La syntaxe de déclaration est la suivante :

```
Dim <Nom de la variable> [As <Type de la variable>]
```

Note :

La déclaration explicite des variables dans le Basic Connect-It correspond à l'utilisation du mot clé *Option Explicit* de Microsoft Visual Basic.

Le nom d'une variable doit respecter les contraintes suivantes :

- Il doit commencer par une lettre majuscule ou minuscule,
- Il ne doit pas comporter plus de 40 caractères,
- Il peut contenir les lettres de A à Z et de a à z, les chiffres de 0 à 9, ainsi que le caractère underscore ("_").

Note :

Les caractères accentués sont autorisés mais leur utilisation est fortement déconseillée.

- Il ne peut être un mot clé réservé. Par exemple, tous les noms de fonctions Basic ou les clauses sont des mots clés réservés.

La clause optionnelle *As* permet de définir le type de la variable déclarée. Le type précise le type d'information stocké dans la variable. Les types disponibles sont par exemple : *String*, *Integer*, *Variant*, ...

Si la clause *As* est omise, la variable est considérée comme étant de type *Variant*.

Déclaration simple

Dans le cas d'une déclaration simple, chaque ligne déclarative concerne une seule variable, comme dans l'exemple suivant :

```
Dim I As Integer
Dim strName As String
Dim dNumber As Double
```

Déclaration combinée

Dans le cas d'une déclaration combinée, chaque ligne déclarative peut concerner un nombre quelconque de variables, comme dans l'exemple suivant :

```
Dim I As Integer, strName As String, dNumber As Double
Dim A, B, C As Integer
```

> **Note :**
>
> Comme il a été décrit précédemment, toute variable dont le type n'est pas précisé est considérée comme étant de type *Variant*. Ainsi, dans la deuxième ligne de l'exemple précédent, les variables *A* et *B* sont de type *Variant* et la variable *C* de type *Integer*.

Types des données

Le tableau ci-dessous récapitule les différents types possibles pour une fonction ou un paramètre :

Type	Signification
Integer	Nombre entier de -32 768 à +32 767.
Long	Nombre entier de -2 147 483 647 à +2 147 483 646.
Single	Nombre à virgule flottante de 4 octets (simple précision).
Double	Nombre à virgule flottante de 8 octets (double précision).
String	Texte pour lequel tous les caractères sont acceptés.
Date	Date ou Date+Heure.
Variant	Type générique pouvant représenter n'importe quel type.

Les types numériques

Le Basic disponible sous Connect-It propose plusieurs types de données numériques : Integer, Long, Single et Double. L'utilisation d'un type de données numérique occupe généralement moins de place en mémoire qu'un *Variant*.

Si vous avez la certitude qu'une variable stockera systématiquement des nombres entiers (par exemple 123) et non des nombres fractionnels (comme 3.14), il est préférable de la déclarer comme un *Integer* ou un *Long*. Les opérations effectuées sur ces types sont plus rapides et moins coûteuses en mémoire que pour les autres types de données. Ces types de donnée sont particulièrement adaptés pour les compteurs utilisés dans les boucles.

Si une variable doit contenir un nombre fractionnel, déclarez-la comme *Single* ou *Double*.

> **Note :**
>
> Les nombres à virgule flottante (*Single* ou *Double*) peuvent être sujets à des erreurs d'arrondis.

Le type String

Si vous avez la certitude qu'une variable stockera systématiquement une chaîne de caractères, déclarez-la comme *String* :

```
Dim MyString As String
```

Vous pouvez alors stocker des chaînes de caractères dans cette variable et manipuler son contenu en utilisant les fonctions dédiées au traitement des chaînes de caractères :

```
MyString = "This is a string"
MyString = Right(MyString,6)
```

Par défaut, une variable de type *String* possède une taille variable. L'espace réservé au stockage des chaînes de caractères augmente ou diminue en fonction de la taille des données affectées à la variable. Il est toutefois possible de déclarer une variable de type *String* dont la taille est fixe, en utilisant la syntaxe suivante :

```
Dim <Nom de la variable> As String * <Taille de la chaîne stockée>
```

L'exemple suivant déclare une variable qui contiendra 20 caractères :

```
Dim MyString As String * 20
```

Si vous stockez dans cette variable une chaîne de moins de 20 caractères, des espaces seront rajoutés en fin de chaîne jusqu'à concurrence de la taille prévue. A l'inverse, si vous stockez une chaîne de plus de 20 caractères, la chaîne sera tronquée à partir du vingtième et unième caractère.

Le type Variant

Le *Variant* est un type générique qui peut se substituer à tous les autres types. Vous n'avez pas à vous soucier d'éventuels problèmes de conversion entre les différents types de données qu'un *Variant* peut représenter. La conversion est réalisée automatiquement, comme dans l'exemple suivant :

```
Dim MyVariant As Variant
MyVariant = "123"
MyVariant = MyVariant - 23
MyVariant = "Top " & MyVariant
```

Bien que la conversion entre les différents types soit automatique, veillez à respecter les règles suivantes :

- Si vous réalisez des opérations arithmétiques sur un *Variant*, celui-ci doit impérativement contenir un nombre, même si celui-ci est représenté par une chaîne de caractères.
- Si une opération de concaténation de chaînes fait intervenir un *Variant*, utilisez l'opérateur & de préférence à l'opérateur +.

Un *Variant* peut également contenir deux valeurs particulières : la valeur vide et la valeur *Null*.

La valeur vide

Avant qu'une valeur soit affectée pour la première fois à une variable de type *Variant*, celle-ci contient la valeur vide. Cette valeur est une valeur particulière, différente de 0, d'une chaîne vide ou encore de la valeur *Null*. Pour tester si un *Variant* contient la valeur vide, utilisez la fonction Basic **IsEmpty()**, comme l'illustre l'exemple suivant :

```
Dim MyFirstVariant As Variant
Dim MySecondVariant As Variant
If IsEmpty(MyFirstVariant) Then MyFirstVariant = 0
MySecondVariant = 0
If IsEmpty(MySecondVariant) Then MySecondVariant = 123
```

Un *Variant* contenant la valeur vide peut être utilisé dans les expressions. Suivant les cas, il sera traîté soit comme de valeur 0, soit comme contenant une chaîne vide. Pour réaffecter la valeur vide à un *Variant*, utilisez le mot clé *Empty*, comme dans l'exemple suivant :

```
Dim MyVariant As Variant
MyVariant = 123
MyVariant = Empty
```

La valeur Null

La valeur *Null* est communément utilisée dans les bases de données pour indiquer une valeur inconnue ou manquante. Cette valeur possède des caractéristiques particulières :

- Les expressions faisant intervenir la valeur *Null* renvoient toujours la valeur *Null*. On parle de propagation de la valeur *Null* dans les expressions. Si une partie de l'expression vaut *Null* alors l'intégralité de l'expression vaut *Null*.
- En règle générale, si un paramètre de fonction a la valeur *Null*, la fonction renvoie la valeur *Null*.

Tableaux de données

Un tableau permet de stocker et de référencer un ensemble de variables par un nom unique et d'utiliser un nombre (un index) pour les identifier de façon unique. Tous les éléments d'un tableau partagent obligatoirement le même type de données. Vous ne pouvez pas créer un tableau contenant à la fois des variables de type *String* et *Double*. Evidemment, cette limitation peut être levée en utilisant des variables de type *Variant*.

Déclaration d'un tableau

Un tableau est un ensemble de variables.

Par convention, les notions suivantes sont présentées comme suit :

- Limite inférieure du tableau : index du premier élément.

> **Note :**
>
> Par défaut la limite inférieure d'un tableau est 0.

- Limite supérieure du tableau : index du dernier élément.

> **Note :**
>
> La limite supérieure d'un tableau ne peut excéder la taille d'un *Long* (2 147 483 646 éléments).

La déclaration d'un tableau est similaire à celle d'une variable :

```
Dim <Nom du tableau>(<Limite supérieure du tableau>) [As <Type des variabl
es contenues dans le tableau>]
```

Exemples :

```
Dim MyFirstArray(30) As String ' 31 elements
Dim MySecondArray(9) As Double ' 10 elements
```

Vous pouvez également préciser la limite inférieure du tableau en utilisant la déclaration suivante :

```
Dim <Nom du tableau>(<Limite inférieure du tableau> To <Limite supérieure
du tableau>) [As <Type des variables contenues dans le tableau>]
```

Exemples :

```
Dim MyFirstArray(1 To 30) As String ' 30 elements
Dim MySecondArray(5 To 9) As Double ' 5 elements
```

Limitations

Les limitations suivantes s'appliquent à la gestion des tableaux dans le Basic Connect-It :

- Les tableaux de taille variable ne sont pas supportés. En particulier, il est impossible de redimensionner un tableau à la voléee.
- Les tableaux à plusieurs dimensions ne sont pas supportés.

Structures de contrôle

Comme leur nom l'indique, les structures de contrôle permettent de contrôler l'exécution d'un programme. Elles sont de deux types :

- Les structures de décision : redirigent et orientent l'exécution d'un programme en fonction de la réalisation de certains critères,
- Les structures de boucles : permettent de répéter l'exécution d'un ensemble d'instructions en fonction de certains critères.

Structures de décision

Une structure de décision réalise l'exécution conditionnelle d'instructions en fonction du résultat d'un test. Les structures de décision disponibles sont les suivantes :

- **If...Then**
- **If...Then...Else...End If**
- **Select Case**

If...Then

Utilisez cette structure pour exécuter conditionnellement une ou plusieurs instructions. La syntaxe de cette structure peut être mono ou multiligne, la syntaxe monoligne ne permettant l'exécution que d'une seule instruction :

```
If <Condition> Then <Instruction>
```

```
If <Condition> Then
<Instructions>
End If
```

La condition est généralement une comparaison, mais toute expression dont le résultat est une valeur numérique peut être utilisée. Cette valeur est alors interprétée comme étant **True** (Vraie) ou **False** (Fausse) par le Basic. **False** correspond à la valeur numérique 0, toute autre valeur étant considérée comme **True**.

Si la condition est évaluée comme **True**, la ou les instructions suivant le mot clé **Then** seront exécutées.

If...Then...Else...End If

Utilisez cette structure pour définir plusieurs blocs d'instructions conditionnelles. Un seul au plus de ces blocs est exécuté (le premier évalué comme **True**).

```
If <Condition1> Then
<Instructions1>
ElseIf <Condition2> Then
<Instructions2>
...
Else
<InstructionsN>
End If
```

La première condition est testée, si le résultat est évalué comme **False**, la deuxième condition est testée et ainsi de suite jusqu'à ce que l'une d'entre elles soit évaluée comme **True**. Le jeu d'instructions situé après le mot clé **Then** de cette condition est alors exécuté.

Le mot clé **Else** est optionnel. Il permet de définir un jeu d'instructions à exécuter si toutes les conditions sont évaluées comme **False**.

Note :

Vous pouvez imbriquer autant de **ElseIf** que vous le souhaitez dans la structure de décision. Néanmoins, si vous comparez systématiquement la même expression à une valeur différente, la syntaxe de la structure de décision peut devenir inutilement complexe et difficile à lire. Nous vous conseillons, dans ce cas, d'utiliser de préférence un structure de décision de type **Select...Case**.

Select...Case

La fonctionnalité de cette structure est identique à celle des structures de décision précédentes, mais le code produit est en général plus lisible. Une structure **Select...Case** effectue un test unique en début de structure et compare le résultat du test aux valeurs de chaque mot clé **Case** dans la structure. S'il y a correspondance, le jeu d'instructions associé au mot clé **Case** est exécuté.

```
Select Case <Test>
[Case <Liste de valeurs 1>
<Instructions1>]
[Case <Liste de valeurs 2>
<Instructions2>]
...
[Case Else
<Instructionsn>]
End Select
```

Chaque liste de valeurs contient une ou plusieurs valeurs, séparées par des virgules. Si plusieurs mots clés **Case** déclarent une ou plusieurs valeurs correspondant au résultat du test, seul le jeu d'instructions associé au premier mot clé **Case** correspondant est exécuté.

Le jeu d'instructions associé au mot clé **Case Else** est exécuté si aucune correspondance n'a été détectée pour les mots clés **Case**.

Structures de boucle

Une structure de boucle permet de répéter l'exécution d'une série d'instructions. Les structures de boucle disponibles sont les suivantes :

- **Do...Loop**
- **For...Next**

Do...Loop

Utilisez cette structure pour exécuter une série d'instructions un nombre de fois non défini. La sortie de la boucle est effectuée lorsqu'une condition est réalisée ou non. Cette condition est une valeur ou une expression qui est évaluée comme **False** (0) ou **True** (différent de 0).

Note :

La sortie de la boucle peut être forcée en utilisant le mot clé **Exit Do** dans les instructions exécutées.

Il existe plusieurs variations de cette structure, mais la plus usitée est la suivante :

```
Do While <Condition>
<Instructions>
Loop
```

Dans ce cas, la condition est évaluée en premier. Si elle est vérifiée (**True**), les instructions sont exécutées et le programme retourne au mot clé **Do While**, teste à nouveau la condition et ainsi de suite. La sortie de boucle est réalisée si la condition est évaluée comme **False**.

L'exemple suivant teste la valeur d'un compteur, incrémenté à chaque passage dans la boucle (ou itération). La sortie de la boucle est réalisée si le compteur contient la valeur 20.

```
Dim iCounter As Integer
iCounter = 0
Do While iCounter < 20
iCounter = iCounter +1
Loop
```

L'exemple suivant reprend l'exemple précédent mais force la sortie de la boucle par un **Exit Do** si le compteur contient la valeur 10.

```
Dim iCounter As Integer
iCounter = 0
Do While iCounter < 20
iCounter = iCounter +1
If iCounter = 10 Then Exit Do
Loop
```

Dans ce type de structure **Do...Loop**, la condition est évaluée avant d'exécuter les instructions. Si vous souhaitez exécuter les instructions puis tester la condition, utilisez la structure **Do...Loop** suivante :

```
Do
<Instructions>
Loop While <Condition>
```

Note :

Ce type de structure garantit au moins une exécution des instructions.

Les deux types de structure **Do...Loop** précédents itèrent tant que la condition est réalisée (**True**). Si vous souhaitez itérer tant que la condition n'est pas réalisée (**False**), utilisez l'une des deux structures suivantes :

```
Do Until <Condition>
<Instructions>
Loop

Do
<Instructions>
Loop Until <Condition>
```

En utilisant ce type de structure, l'exemple précédent peut s'écrire :

```
Dim iCounter As Integer
iCounter = 0
Do Until iCounter = 20
iCounter = iCounter +1
Loop
```

For...Next

Utilisez cette structure pour exécuter une série d'instructions un nombre de fois déterminé. A l'inverse des structures **Do...Loop**, une boucle **For...Next** utilise une variable appelée compteur dont la valeur augmente ou diminue à chaque itération.

Note :

La sortie de la boucle peut être forcée en utilisant le mot clé **Exit For** dans les instructions exécutées.

```
For <Compteur> = <Valeur initiale> To <Valeur finale> [Step <Incrément>]
<Instructions>
Next [<Compteur>]
```

IMPORTANT :

Les arguments **Compteur**, **Valeur initiale**, **Valeur finale** et **Incrément** sont tous représentés par des valeurs numériques.

Note :

Incrément peut être une valeur positive ou négative. Si elle est positive la **Valeur initiale** doit être inférieure ou égale à la **Valeur finale** pour que les instructions soient exécutées. Si elle est négative, la **Valeur initiale** doit être supérieure ou égale à la **Valeur finale** pour que les instructions soient exécutées. Si l'**Incrément** n'est pas précisé, sa valeur par défaut est 1.

Lors de l'exécution d'une boucle **For...Next**, les opérations suivantes sont réalisées :

1. Le compteur est initialisé et stocke la valeur initiale,
2. Le Basic teste si la valeur du compteur est supérieure à la valeur finale. Si tel est le cas, le programme sort de la boucle.

   Note :

   Si l'incrément est négatif, le Basic teste si la valeur du compteur est inférieure à la valeur finale.

3. Les instructions sont exécutées,
4. Le compteur est incrémenté de 1 ou de la valeur spécifiée par l'incrément,
5. Les opérations 2 à 4 sont répétées.

L'exemple suivant effectue la somme des nombres pairs jusqu'à 1000 :

```
Dim iCounter As Integer, lSum As Long
For iCounter = 0 To 1000 Step 2
lSum = lSum + iCounter
Next
```

L'exemple suivant reprend l'exemple précédent mais force la sortie de la boucle par un **Exit For** si le compteur contient la valeur 500.

```
Dim iCounter As Integer, lSum As Long
For iCounter = 0 To 1000 Step 2
lSum = lSum + iCounter
If iCounter = 500 Then Exit For
Next
```

Opérateurs

Les opérateurs sont des symboles qui permettent de réaliser des opérations simples (addition, multiplication, ...) entre des variables et de les évaluer ou les comparer. On distingue plusieurs types d'opérateurs :

- Les opérateurs d'affectation,

- Les opérateurs de calcul,
- Les opérateurs relationnels (également appelés opérateurs de comparaison),
- Les opérateurs logiques.

Les opérateurs d'affectation

Ce type d'opérateur permet d'affecter une valeur à une variable. Le Basic Connect-It utilise une seule variable d'affectation, le signe "=". La syntaxe d'affectation est la suivante :

```
<Variable> = <Valeur>
```

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable, ou de réaliser des opérations mathématiques simples entre deux expressions.

L'opérateur +

Cet opérateur permet d'effectuer la somme de deux valeurs. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> + <Expression 2>
```

Note :

Cet opérateur est utilisé tant pour effectuer la somme de deux nombres que pour concaténer des chaînes. Pour lever toute ambiguité, nous vous conseillons de réserver l'utilisation de cet opérateur à des sommes, et d'utiliser l'opérateur & pour concaténer des chaînes.

L'opérateur -

Cet opérateur permet d'effectuer la différence entre deux valeurs ou de signer négativement (opérateur monadique) une valeur. L'opérateur possède donc deux syntaxes :

```
<Résultat> = <Expression 1> - <Expression 2>
```

ou

```
- <Expression>
```

L'opérateur *

Cet opérateur permet d'effectuer la multiplication de deux valeurs. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> * <Expression 2>
```

L'opérateur /

Cet opérateur permet d'effectuer la division de deux valeurs. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> / <Expression 2>
```

L'opérateur ^

Cet opérateur permet d'élever une valeur à la puissance d'un exposant. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> ^ <Expression 2>
```

Note :

Dans cette syntaxe l'expression 1 ne peut être négative que si l'expression 2 (l'exposant) est une valeur entière. Lorsqu'une expression effecute plusieurs opérations d'exposants en série, le Basic les interprète logiquement, de gauche à droite.

L'opérateur Mod

Cet opérateur permet de calculer le reste de la division euclidienne de deux valeurs. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> Mod <Expression 2>
```

Note :

Les nombres à virgule flottante sont systématiquement arrondis à l'entier le plus proche.

L'exemple suivant renvoie la valeur 4 (6.8 est arrondi à l'entier le plus proche lors du calcul, soit 7):

```
Dim iValue As Integer
iValue = 25 Mod 6.8
```

Les opérateurs relationnels

Les opérateurs relationnels permettent de comparer des valeurs. Le tableau ci-dessous propose une vue d'ensemble des opérateurs relationnels :

Opérateur	Dénomination	Description	Syntaxe
=	Opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	`<Expression 1> = <Expression 2>`

Opérateur	Dénomination	Description	Syntaxe
<	Opérateur d'infériorité stricte	Vérifie qu'une valeur est strictement inférieure à une autre	`<Expression 1> < <Expression 2>`
<=	Opérateur d'infériorité	Vérifie qu'une valeur est inférieure ou égale à une autre	`<Expression 1> <= <Expression 2>`
>	Opérateur de supériorité stricte	Vérifie qu'une valeur est strictement supérieure à une autre	`<Expression 1> > <Expression 2>`
>=	Opérateur de supériorité	Vérifie qu'une valeur est supérieure ou égale à une autre	`<Expression 1> >= <Expression 2>`
<>	Opérateur de différence	Vérifie qu'une valeur est différente d'une autre	`<Expression 1> <> <Expression 2>`

Les opérateurs logiques

Les opérateurs logiques permettent de vérifier la véracité de plusieurs conditions.

L'opérateur And

Cet opérateur effectue un ET logique (les deux conditions doivent être vérifiées) entre deux expressions. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> And <Expression 2>
```

Si chaque expression est évaluée comme vraie (*True*), le résultat est vrai (*True*). Si l'une des deux expressions est évaluée comme fausse (*False*), le résultat est évalué comme faux (*False*).

L'opérateur Or

Cet opérateur effectue un OU logique (une des deux conditions doit être vérifiée) entre deux expressions. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> Or <Expression 2>
```

Si l'une ou les deux expressions sont évaluées comme vraies (*True*), le résultat est vrai (*True*).

L'opérateur Xor

Cet opérateur effectue un OU exclusif (une seule des deux conditions doit être vérifiée) entre deux expressions. La syntaxe est la suivante :

```
<Résultat> = <Expression 1> Xor <Expression 2>
```

Si une seule des deux expressions est évaluée comme vraie (*True*), le résultat est vrai (*True*).

L'opérateur Not

Cet opérateur effectue la négation logique d'une expression. La syntaxe est la suivante :

```
<Résultat> = Not <Expression 1>
```

Si l'expression est évaluée comme vraie (*True*), le résultat est faux (*False*). Si l'expression est évaluée comme fausse (*False*), le résultat est vrai (*True*).

Priorité des opérateurs

Lorsque plusieurs opérateurs sont associés, l'ordre de priorité suivant est respecté dans l'évaluation des expressions. La liste ordonnée suivante classe les opérateurs dans un ordre de priorité décroissant :

1 ()
2 ^
3 -, +
4 /, *
5 Mod
6 =, >, <, <=, >=
7 Not
8 And
9 Or
10 Xor

Gestion de fichiers

Le Basic Connect-It permet de manipuler des fichiers de façon simple. Les opérations les plus usuelles (lecture, écriture,...) sont disponibles en standard.

Rappel préliminaire sur les fichiers

Un fichier est la façon dont un programme voit un objet externe. Il s'agit d'une collection d'enregistrements logiques, éventuellement structurée, sur laquelle le programme peut exécuter un ensemble d'opérations élémentaires (lecture, écriture, ...). Un enregistrement logique représente l'ensemble minimum de données qui peut être manipulé par une seule opération élémentaire.

Le Basic Connect-It gère uniquement les fichiers dits séquentiels. Dans un fichier séquentiel, les opérations se résument essentiellement en la lecture de l'enregistrement suivant ou l'écriture d'un nouvel enregistrement en fin du fichier séquentiel. Il n'est pas possible de réaliser simultanément lecture et écriture des enregistrements.

En lecture, le fichier séquentiel est initialement positionné sur le premier enregistrement logique. Chaque opération de lecture transfère un enregistrement dans une zone interne (en général une variable) du programme et positionne le fichier sur l'enregistrement suivant. Une opération permet de déterminer s'il reste des enregistrements à lire (clause **EOF** : End Of File).

En écriture, le fichier séquentiel peut être initialement vide ou positionné après le dernier enregistrement du fichier. Chaque opération d'écriture transfère des données stockées dans une zone interne (en général une variable) du programme, dans un enregistrement du fichier et positionne le fichier après cet enregistrement.

Note :

Une des caractéristiques essentielles d'un fichier séquentiel est que les enregistrements sont lus dans l'ordre où ils ont été écrits.

Ouvrir et fermer des fichiers

La clause Open

Il s'agit de la clause de base pour toute manipulation de fichier. Elle permet d'ouvrir un fichier, que ce soit pour le lire, le créer, ou y écrire. La syntaxe est la suivante :

```
Open <Chemin du fichier> For <Mode> [Access <Type d'accès>] As [#]<Numéro de fichier>
```

Les paramètres de cette clause sont détaillés dans le tableau suivant :

Paramètre	Description
<Chemin du fichier>	Chaîne de caractères spécifiant le fichier concerné par l'opération. Cette chaîne peut contenir le chemin complet du fichier.

Paramètre	Description
<Mode>	Précise le mode de traitement du fichier. Ce paramètre peut contenir l'une des valeurs suivantes : ■ *Input* : le fichier est ouvert en lecture. ■ *Output* : le fichier est ouvert en écriture. Si le fichier existe et possède du contenu, celui-ci est écrasé. ■ *Append* : le fichier est ouvert en écriture. Si le fichier existe et possède du contenu, le nouveau contenu est ajouté en fin de fichier.
<Type d'accès>	Précise les opérations réalisables sur un fichier ouvert. Si le fichier est ouvert par un autre processus et que le type d'accès précisé n'est pas autorisé, la commande d'ouverture de fichier échoue. Ce paramètre peut contenir l'une des valeurs suivantes : ■ *Read* : le fichier est ouvert en lecture seule ■ *Write* : le fichier est ouvert en écriture seule ■ *Read Write* : le fichier est ouvert en lecture-écriture. Ce type d'accès n'est disponible que pour des accès en mode *Append*.
<Numéro de fichier>	Identifie le fichier par un numéro unique compris entre 1 et 511. La fonction **FreeFile()** permet de déterminer le prochain numéro de fichier disponible.

 Note :

Gardez en mémoire les points suivants :

- Tout fichier doit être ouvert par la clause **Open** avant de lire ou d'écrire des informations dans ce fichier.
- En mode *Append*, *Binary* ou *Output*, si le fichier référencé n'existe pas, il est créé.
- En mode *Binary* ou *Input*, vous pouvez ouvrir un fichier en utilisant un numéro différent sans fermer le fichier au préalable. En mode *Append* ou *Output*, vous devez impérativement fermer un fichier avant de l'ouvrir à nouveau sous un numéro différent.

La clause Close

Cette clause permet de fermer un fichier préalablement ouvert au moyen de la clause **Open()**. La syntaxe est la suivante :

```
Close [<Liste de fichiers>]
```

L'argument optionnel **<Liste de fichiers>** peut contenir un ou plusieurs numéros de fichiers. La syntaxe de cet argument optionnel est la suivante :

```
[[#]<Numéro de fichier>][,[#]<Numéro de fichier>]...
```

Note :

Si vous omettez le paramètre de cette clause, tous les fichiers actifs ouverts par une clause **Open()** sont fermés.

Lire des données d'un fichier

Deux clauses sont disponibles pour la lecture des données d'un fichier. L'utilisation de l'une ou l'autre de ces clauses dépend du mode d'accès spécifié pour le fichier. Les deux clauses sont les suivantes :

- **Input**
- **Line Input**

La clause Input

Cette clause est utilisée pour lire un nombre déterminé de caractères à partir d'un fichier ouvert en mode *Binary* ou *Input*. La syntaxe de cette clause est la suivante :

```
Input (<Nombre de caractères à lire>,[#]<Numéro de fichier>)
```

La clause Line Input

Cette clause est utilisée pour lire une ligne de données d'un fichier séquentiel, et la stocker dans une variable de type *String* ou *Variant*. La syntaxe de cette clause est la suivante :

```
Line Input #<Numéro de fichier>, <Nom de la variable>
```

IMPORTANT :

La clause lit les caractères un par un jusqu'à ce qu'un retour chariot ou un ensemble retour chariot - fin de ligne soit rencontré.

Ecrire des données dans un fichier

Une seule clause, **Print**, permet l'écriture de données dans un fichier. La syntaxe de cette clause est la suivante :

```
Print #<Numéro de fichier>, [<Données>]
```

Note :

La clause **Print** écrit les données dans un fichier sur une seule et même ligne, jusqu'à ce qu'un caractère de saut de ligne (code ASCII 10) soit rencontré. Par exemple :

```
Open "c:\test.txt" For Output As #1
Print #1, "This is a test" & Chr(10)
Print #1, "And now we can close the file..." & Chr(10)
Close #1
```

2 Champ d'application des fonctions

Les fonctions décrites dans ce document sont toutes utilisables dans les fenêtres de script de Connect-It.

3 Conventions

Ce chapitre contient des informations sur les conventions d'écriture utilisées dans ce document et sur le format de certaines constantes.

Conventions d'écriture

La syntaxe des fonctions et des exemples proposés respecte les conventions d'écriture suivantes :

[]	Ces crochets encadrent un paramètre optionnel. Ne les tapez pas dans votre commande. Exception : dans les scripts Basic, lorsque les crochets encadrent le chemin d'accès à des données de la base, ils doivent figurer dans le script, comme le montre l'exemple ci-dessous : `[Lien.Lien.Champ]`
<>	Ces crochets encadrent un paramètre décrit en langage clair. Ne les tapez pas dans votre commande et remplacez le texte qu'ils encadrent par l'information qui doit y figurer.
{}	Ces accolades encadrent la définition d'un noeud ou d'un bloc de script multilignes pour une propriété.

|	La barre verticale sépare les paramètres possibles qui figurent dans les accolades.

Les mises en forme suivantes ont des significations particulières :

`Police fixe`	Commande DOS, paramètre de fonction ou formatage de données.
`Exemple`	Exemple de code ou de commande.
...	Portion de code ou de commande omise.
Nom d'objet	Les noms de champs, d'onglets, de menus, de fichiers sont en caractères gras.
Note : Note	Note importante.

Format des constantes de type Date+Heure dans les scripts

Les dates référencées dans les scripts sont exprimées au format international, indépendamment des options d'affichage spécifiées par l'utilisateur :

`yyyy/mm/dd hh:mm:ss`

Exemple :

```
RetVal="1998/07/12 13:05:00"
```

Note :

Le tiret ("-") peut également être utilisé comme séparateur de date.

Date Basic et Date Unix

Les dates sont exprimées différemment en Basic et Unix :

- En Basic, une date peut être exprimée au format international ou sous la forme d'un nombre à virgule flottante (type "Double"). Dans ce dernier cas, la partie entière de ce nombre représente le nombre de jours écoulés depuis le 30/12/1899 à 0:00, la partie décimale représente la fraction écoulée dans le jour courant.
- Sous Unix, les dates sont exprimées sous la forme d'un entier long (type "Long" 32 bits) qui représente le nombre de secondes écoulées depuis le 01/01/1970 à 0:00, indépendamment d'un quelconque fuseau horaire (heure UTC).

Format des constantes de type Durée

Dans les scripts, les durées sont stockées et exprimées en secondes. Par exemple, pour fixer la valeur par défaut d'un champ de type "Durée" à 3 jours, vous devez utiliser le script suivant :

```
RetVal=259200
```

De même, les fonctions qui calculent ou traitent une durée fournissent un résultat en secondes.

 Note :

Dans les calculs financiers, Connect-It tient compte des simplifications habituellement usitées. Dans ce cas seulement, une année vaut 12 mois et un mois vaut 30 jours (d'où : 1 année = 360 jours).

4 Définitions

Ce chapitre regroupe les définitions de quelques termes essentiels.

Définition d'une fonction

Une fonction est un programme qui effectue des opérations et renvoie à l'utilisateur une valeur, appelée "valeur de retour" ou "code de retour".

Voici un exemple de syntaxe d'appel d'une fonction par le Basic interne de Connect-It :

```
PifIgnoreDocumentMapping(strMsg As String) As Long
```

Définition d'un code d'erreur

Lorsque l'exécution d'une fonction échoue, un code d'erreur est renvoyé.

La description et le code de la dernière erreur peuvent être retrouvés respectivement au moyen des fonctions **Err.Description** et **Err.Number**.

Vous pouvez déclencher volontairement un message d'erreur en utilisant la fonction **Err.Raise** dont la syntaxe est la suivante :

```
Err.Raise (<Numéro de l'erreur>, <Message d'erreur>)
```

Le tableau ci-dessous répertorie les codes d'erreur les plus fréquents :

Code d'erreur	Signification
12001	Erreur indéfinie
12002	Mauvais paramètre pour une fonction
12003	Handle invalide ou objet détruit
12004	Plus de données disponible. Cette erreur arrive classiquement lors de l'exécution de requêtes. Quand le résultat de la requête ne renvoie aucune donnée, cette erreur est déclenchée.
12006	Valeur non valide (type incorrect pour un paramètre, etc.)
12009	Fonction obsolète ou non implémentée

5 Typage des fonctions et des paramètres de fonctions

Liste des types

Le tableau ci-dessous récapitule les différents types possibles pour une fonction ou un paramètre :

Type	Signification
Integer	Nombre entier de -32 768 à +32 767.
Long	Nombre entier de -2 147 483 647 à +2 147 483 646.
Single	Nombre à virgule flottante de 4 octets (simple précision).
Double	Nombre à virgule flottante de 8 octets (double précision).
String	Texte pour lequel tous les caractères sont acceptés.
Date	Date ou Date+Heure.

Type	Signification
Variant	Type générique pouvant représenter n'importe quel type.

Type d'une fonction

Le type d'une fonction correspond au type de la valeur retournée par la fonction. Nous vous invitons à faire particulièrement attention à cette information car elle peut être à l'origine d'erreurs de compilation et d'exécution de vos programmes.

Type d'un paramètre

Les paramètres utilisés dans les fonctions possèdent également un type que vous devez impérativement respecter pour la bonne exécution de la fonction. Dans la syntaxe des fonctions, les paramètres sont préfixés en fonction de leur type. Le tableau ci-dessous résume les différents types utilisés et le préfixe qui leur est associé :

Type	Préfixe utilisé dans la syntaxe Basic
Integer	"i"
Long	"l"
Double	"d"
String	"str"
Date	"dt"
Variant	"v"

6 Exemples de script

Cette section vous présente des exemples de scripts de mapping classés en fonction des éléments qu'ils utilisent.

Fonctions Basic

If, Then, Else, Else If, End If

Syntaxe

If <condition> **Then**
<Instructions>
Else If <condition> **Then**
<Instructions>
Else
<Instructions>
End If

Note :

A propos des champs logiques (booléen) :

Les champs logiques sont représentés sous la forme d'entier 8-bits. La valeur "vrai", en Basic, est égale à -1.

Certains scripts portant sur des champs logiques peuvent poser problème :

```
if [logicalfield] = true Then
```

Si la valeur "vrai" définie pour votre base de données est 1 et la valeur "faux" est 0, alors, pour ce script, la valeur retournée sera égale à 1, donc toujours fausse au sens Basic.

Exemple

```
Dim strVal As String
(...)
If strVal = "" Then
RetVal = "Empty"
ElseIf strVal = "Default" Then
RetVal = "Default"
Else
RetVal = "Unknown"
End If
```

Ce script retourne la valeur :

- *Empty* si le champ texte d'un document produit ne contient aucune information.
- *Default* si le champ texte d'un document produit contient l'information *default*.
- *Unknown* si le champ texte d'un document produit contient tout autre information.

For Loop

Cette fonction permet de créer une boucle.

Syntaxe

For <variable du compteur> = <début> **to** <fin>

<Instructions>

Next

Exemple

```
For i=0 To 10 Step 2
PifLogInfoMsg(i)
Next
```

Ce script permet de retourner la valeur de *i* dans le journal des documents.

La visualisation dans le journal des documents est la suivante :

```
0
2
4
6
8
10
```

While Loop

Cette instruction permet de créer une boucle.

Syntaxe

While loop

While <conditions>

<instructions>

WEnd

Exemple

```
Dim i As Integer
i = 0
While i < 10
i = i + 2
PifLogInfoMsg(i)
WEnd
```

Ce script retourne la valeur de *i* dans le journal des documents tant que cette valeur est inférieure à 10.

La visualisation dans le journal des documents est la suivante :

```
0
2
4
6
8
10
```

Return

Dans un script, si les conditions définies avant cette fonction ne sont pas respectées, le reste du script est ignoré.

Syntaxe

<conditions>

Return

<conditions>

Exemple

```
If [MacAddress] = "" And [IPAddress] = "" Then
PifIgnoreNodemapping
Return
End If

If [MacAdress] <> "" Then
RetVal = [MacAdress]
Else
RetVal = [IPAdress]
End If
```

Ce script teste si les champs **MacAdress** et **IPAdress** d'un document produit n'ont pas une valeur vide. Si cette condition est remplie :

- le noeud courant est ignoré
- la fin du script n'est pas exécutée

Select

Cette fonction permet d'exécuter un bloc d'instructions en fonction de la valeur d'une variable.

Syntaxe

Select Case <variable à tester>

Case <variable 1>

Bloc d'instructions

Case <variable 2>

Bloc d'instructions

Case <variable 3>

Bloc d'instructions

...

Case <variable n>

Bloc d'instructions

Case Else

End Select

Exemple

```
Select Case [seStatus]
Case 0
RetVal = "Opened"
Case 1
RetVal = "Closed"
Case Else
RetVal = "Unknown status"
End Select
```

Dans cet exemple :

- Le champ **seStatus** du document source correspond à l'état d'un dossier de support.
- L'état d'un dossier de support est :
  - 0 = dossier de support ouvert
  - 1 = dossier de support fermé

Ce script associe la chaîne de caractères décrivant l'état du dossier de support à la valeur numérique du champ source. Si l'état n'est pas connu, la valeur *Unknown Status* est retournée.

Fonctions Pif

Les fonctions de type **PIF** ont été spécialement développées pour les scripts de mappings de Connect-It.

PifIgnoreDocumentMapping

Cette fonction permet d'ignorer le traitement d'un document.

Syntaxe

<conditions>

PifIgnoreDocumentMapping("<message>")

<conditions>

("message") vous permet d'afficher un message d'erreur dans le journal des documents pour l'élément ignoré.

La spécification d'une fonction *retval* implique que la fonction de type *PifIgnore* s'exécute sur un champ choisi comme clé de réconciliation.

Exemple

```
If [MacAddress] = "" Then
PifIgnoreDocumentMapping("Missing MAcAdress")
End If
RetVal = [MacAddress]
```

On utilise le champ **MacAddress** comme clé de réconciliation. Si ce champ ne contient aucune valeur, le document est ignoré. Le message *champ MacAddres manquant* est visible dans le journal des documents.

PifRejectDocumentMapping

Cette fonction permet de rejeter un document source et de ne pas l'envoyer vers le connecteur de destination.

Elle s'applique à n'importe quel élément du document :

- noeud racine
- structure
- collection
- champ

Syntaxe

<instructions>

PifRejectDocumentMapping("message")

<instructions>

("message") vous permet d'afficher un message d'erreur dans le journal des documents pour l'élément ignoré.

La spécification d'une fonction *retval* implique que la fonction de type *PifReject* s'exécute sur un champ choisi comme clé de réconciliation.

Exemple

```
If [MacAdress] = "" Then
PifRejectDocumentMapping("Missing MAcAdress")
End If
RetVal = [MacAdress]
```

On utilise le champ **MacAddress** comme clé de réconciliation. Si ce champ ne contient aucune valeur, le document est rejeté. Le message *champ MacAddres manquant* est visible dans le journal des documents.

PifIgnoreNodeMapping

Cette fonction permet d'ignorer n'importe quel élément d'un type de document.

Cet élément peut être :

- Le noeud racine du document
- Une structure
- Une collection
- Un champ

Le comportement de la fonction *PifIgnoreNodeMapping* est différent selon qu'elle porte sur une collection ou non.

Si cette instruction porte sur une collection, seul le membre courant de la collection est ignorée. Si vous souhaitez ignorer tous les membres de la collection, utilisez l'instruction *PifIgnoreCollectionMapping*.

Syntaxe

```
(...)
PifIgnoreNodeMapping("Message")
(...)
```

("message") vous permet d'afficher un message d'erreur dans le journal des documents pour l'élément ignoré.

Exemple

```
If [MacAdress] = "" Then
PifIgnoreNodeMapping
End If
RetVal = [MacAdress]
```

Ce script permet d'éviter une mise à jour avec une chaîne vide si le champ ou la structure contenant le champ **MACaddress** est vide. Si le champ est renseigné, alors la mise à jour est effectuée.

```
If Left([Software.Name], 7)  = "Windows" Then
PifIgnoreNodeMapping
ElseIf Left([Software.Name], 5) = "SunOS" Then
PifIgnoreDocumentMapping
End If
```

Ce script permet d'ignorer le membre d'une collection si le champ **Sofware.Name** de ce membre a pour valeur *Windows* ou *SunOS*.

PifIgnoreCollectionMapping

Cette fonction permet d'ignorer une collection d'un type de document produit lors d'un mapping de collection à collection.

Pour plus d'informations sur le mapping collection à collection, consultez le manuel de référence Connect-It - Utilisation, chapitre *Mise en place d'un scénario d'intégration*.

Syntaxe

<instructions>

PifIgnoreCollectionMapping

<instructions>

Exemple

```
Dim i As Integer
Dim iCount As Integer
Count = PifGetItemCount("Logs")
For i=0 To iCount - 1
If [Logs(i).LogType] = 1 Then
Return
End If
Next
PifIgnoreCollectionMapping
```

Dans ce script, pour un bilan de traitement, tous les membres de la collection **logs** sont ignorés si il n'y a pas de message d'erreur.

Si le document ne contient aucune erreur, il n'est pas nécessaire de procéder à un tel script. Le champ **ErrorNumber** contient le nombre d'erreurs associées au document.

Le script précédent peut être remplacé par le suivant :

```
If [ErrorNumber] = 0 Then
PifIgnoreCollectionMapping
End If
```

Collections

Dans cette section, vous trouverez différents exemples de scripts portant sur le traitement des collections.

Créer des membres dans une collection à partir d'une liste de valeurs

Cette section vous présente un exemple de script permettant de créer un membre dans une collection donnée à partir d'un liste de valeurs d'un document source.

Dans cet exemple :

- Le champ source **Software** contient une liste de valeurs.

- Les valeurs sont délimitées par un séparateur donné.

Le script :

- Extrait le nom des logiciels un par un.
- Crée un membre dans la collection destination **SoftInstalled**.
- Renseigne l'élément **Name** avec le nom du logiciel extrait.

```
Dim iCount  As Integer
Dim iIndex  As Integer
Dim strSoft As String
Dim lDummy  As Long
Dim strPath As String

' Décompte du nombre de valeurs dans le champ source "Software"
' les noms des logiciels sont délimités par le séparateur virgule (','), p
ar exemple "Excel, Connect-It,
' Asset Manager"
iCount = CountValues([Software], ",")

' Boucle sur tous les éléments de la liste de facon a les extraire un a un
.
For iIndex = 0 To iCount - 1

strSoft = GetListItem([Software], ",", iIndex+1)

' Suppresion des espaces autour du nom du logiciel
strSoft = Trim(strSoft)

' Création du chemin de la collection destination à partir de l'élément ra
cine.
' Par exemple, pour le 3eme logiciel de la source, le chemin "SoftInstalle
d(3).Name" est cree
strPath = "SoftInstalled" (& iIndex & ").Name"

' Affectation de la valeur courante du logiciel de type chaîne de caractèr
es à ce chemin en utilisant
' la fonction PifSetStringVal.
' La fonction PifSetStringVal retournant un code d'erreur si le chemin n'e
st pas valide, il est
' nécessaire d'affecter dans une variable la valeur de retour de la foncti
on. La fonction ne serait
' pas appliquée dans le cas contraire.
lDummy = PifSetStringVal(strPath, strSoft)

Next iIndex
```

Ce script de mapping peut être appliqué sur un élément quelconque du type de document destination.

Pour une meilleur lisibilité du mapping, nous vous conseillons d'effectuer ce mapping sur la collection à laquelle des membres doivent être ajoutés.

Avertissement :

L'élément indiqué par son chemin dans l'appel à la fonction Basic **PifSetStringVal** doit être présent dans le type de document destination. Dans l'exemple présent, l'élément **Name** de la collection **SoftInstalled** doit être ajouté par l'utilisateur dans le type de document consommé.

Concaténer des membres d'une collection dans un champ

Dans cet exemple :

- Le document source contient une collection de valeurs.
- Les éléments de cette collection sont mappés à un champ du type de document destination.

La source contient la collection des logiciels installés sur une machine. Les différents noms de logiciels doivent être écrits dans un champ contenant la liste des logiciels séparés par une virgule (',').

```
Dim iCollectionCount As Integer
iCollectionCount = PifGetItemCount("SoftInstalled")

Dim strList As String
Dim iItem As Integer

Pour chaque élément de la collection, récupérer le nom du logiciel (élémen
t "Name" de la collection "SoftInstalled") et le concatener avec la liste
courante.
For iItem = 0 to iCollectionCount - 1

' Ajouter le séparateur de nom si la liste n'est pas vide
If strList = "" Then
strList = strList & ", "

' Ajouter le nom du logiciel à la liste courante.
' Remarquez qu'il est possible d'utiliser directement une variable pour in
diquer le numéro
' d'un membre dans une collection. Par exemple, si la variable de iITem va
ut 3, le chemin
' [SoftInstalled(3).Name] sera automatiquement créé à partir de la valeur
de iItem.
  strList = strList & ", " [SoftInstalled(iItem).Name]
Next iItem

' Affecter la variable strList à l'élément destination
RetVal = strList
```

Mapper plusieurs champs dans une collection

Dans cet exemple :

- Le document source contient plusieurs champs distincts.

  Ici, les champs *Address1* et *Adress2* comportent les 2 adresses possibles d'un client.
- La valeur de ces champs doit être associée à un membre de la collection destination.

  Ici, la collection *Adress*.

Par exemple :

Il faut donc :

- Créer deux membres dans la collection destination et les associer aux champs "Adress1" et "Adress2".
- Utiliser la fonctionnalité de duplication de collection :
  1. Ajouter la collection *Adress* dans le type de document destination.
  2. Dupliquer cette collection.

     La collection *Adress#1* apparaît dans le type de document destination.
  3. Les scripts de mappings [Adress1] et [Adress2] sont à appliquer respectivement sur les champs *Adress.Adress* et *Adress#1.Adress*

Ignorer certains membres d'une collection dans un mapping collection a collection

Pour ignorer certains membres d'une collection, il vous faut utiliser les instructions *PifIgnoreCollectionMapping* et *PifIgnoreNodeMapping*.

Pour plus de renseignements sur ces instructions, consultez la section PifIgnoreCollectionMapping [page 47].

Script portant sur un connecteur non inclus dans un mapping

L'exemple qui suit décrit l'intégration dans un scénario (portant sur la réplication de données entre une base de données et une base de données ServiceCenter) d'une base de données Asset Manager. Le script importe un employé. Durant l'import, le script vérifie si l'employé existe dans la base Asset Manager et adapte le mapping en conséquence.

1. Ajoutez un connecteur Asset Manager à votre scénario. Ce connecteur n'a pas besoin d'être lié à une boîte de mapping ou un autre connecteur, seul son intitulé est important (champ **Nom du connecteur** dans l'assistant de configuration du connecteur) car il sera utilisé dans le script. Ici, le connecteur est appelé Asset Management.
2. Créez un nouveau type de document produit par le connecteur Asset Manager. Sélectionnez la table des personnes (**amEmplDept**) et appelez le

type de document produit (champ **Type de document**) **amEmplDeptForMapping**. Ce nom sera utilisé dans le script.

Note :

Si vous définissez des clauses WHERE ou ORDER BY, elles ne sont pas prises en compte dans le script d'exemple.

3 Dans la boîte de mapping, renseignez le champ script comme suit :

```
dim hQuery as long
dim iRc as long

hQuery = pifNewQueryFromFmtName("Asset Management", "amEmplDeptForMappi
ng", "Name like 'A%'")

Dim strValue as string

while (iRc = 0)
iRc = pifQueryNext(hQuery)
if iRc = 0 then
strValue = pifQueryGetStringVal(hQuery, "Name")
piflogInfoMsg strValue
end if
wend

iRc = pifQueryClose(hQuery)
```

Syntaxe de la fonction pifNewQueryFromFmtName

Cette fonction crée une requête sur un type de document préalablement défini dans la liste des documents produits par une ressource.

Les paramètres de la fonction sont les suivants :

- **strCntrName** : Ce paramètre contient le nom de la ressource (celle sur laquelle la requête est effectuée).
- **strFmtName** : Ce paramètre contient le nom du type de document (préalablement défini comme type de document produit).
- **strLayer** : Ce paramètre contient les directives de production (par exemple, une clause WHERE).

La fonction retourne un descripteur (ici, le paramètre **hQuery**). Ce descripteur doit être passé comme paramètre de la fonction **PifQueryNext** afin de parcourir la liste des enregistrements retournés.

Les données du document courant peuvent être ensuite récupérées en utilisant une des fonctions suivantes (selon le type de champ examiné) :

- pifQueryGetStringVal
- pifQueryGetDateVal

- pifQueryGetDoubleVal
- pifQueryGetLongVal
- pifQueryGetIntVal

Chacune de ces fonctions comporte deux paramètres :

- Le descripteur (hQuery) de la requête à utiliser (entier long 32 bits)
- Chemin de l'élément pour lequel récupérer une valeur. Ce chemin ne doit pas contenir le nom de l'élément racine du type de document (**amEmplDept** dans cet exemple).

  Dans cet exemple, la fonction retourne le nom de la personne :

```
strValue = pifQueryGetStringVal(hQuery, "Name")
```

Directives de production de la fonction pifNewQueryFromFmtName

La fonction **pifNewQueryFromFmtName** est paramétrée avec des paramètres simples. Vous pouvez néanmoins définir des requêtes plus complexes au format XML.

Les directives de production peuvent être sous forme XML, en respectant la syntaxe suivante :

```
strLayer =             "<Directives>"
strLayer = strLayer +    "<Where>Name = 'Taltek'</Where>"
strLayer = strLayer +    "<OrderBy>BarCode</OrderBy>"
strLayer = strLayer +       "<Where Path='ItemsUsed'>AssetTag like 'A%'</Wh
ere>"
strLayer = strLayer + "</Directives>"

hQuery = pifNewQueryFromFmtName("Asset Management", "amEmplDeptForMapping"
, strLayer)
```

Syntaxe XML

Les caractères &, < et > ne sont pas autorisés. Vous devez les remplacer respectivement par &, < et >. La fonction Basic **GetXmlElementValue** prend en charge le remplacement de ces caractères.

Par exemple :

```
strLayer = strLayer + "<Where Path='ItemsUsed'>" + GetXmlElementValue("Ass
etTag like 'A%'") + "</Where>"
```

Requête sur des champs comportant un point ou une virgule

Dans l'exemple commenté suivant, une requête porte sur les éléments **mac.address** et **logical.name** d'un scénario HP Network Discovery - ServiceCenter. Le script vérifie l'adresse MAC fournie par le connecteur IND

avant de lui attribuer une clé de réconciliation. Si l'adresse MAC est validée, les informations du champ **logical.name** sont récupérées en lieu et place de celles du champ **mac.address**.

```
dim hQuery as long
dim iRc as long
dim strQuery as String

strQuery = "mac.address = " & chr(34) & [MACAddress] & chr(34)
"adresse MAC dans le document PDI

hQuery = pifNewQueryFromFmtName("ServiceCenter", "pc1", strQuery)
"pc1 est le document produit pour la table des ordinateurs de ServiceCente
r

Dim strValue as String
strValue = [MACAddress]
"strValue par défaut

iRc = pifQueryNext(hQuery)
if iRc = 0 then          "
"requête aboutie car iRc=0

strValue = pifQueryGetStringVal(hQuery, "'logical.name'")
"Les guillemets simples définissent le paramètre logical.name comme un cha
mp et non comme un chemin

pifLogWarningMsg("Matched Asset using query: " & strQuery)
"écriture au journal des documents

pifLogWarningMsg("Updating Asset " & strValue)
"strValue n'est pas inscrite au journal des documents
else
pifLogWarningMsg("Could not locate existing asset using MAC address " & [M
acAddress])
end if

iRc = pifQueryClose(hQuery)

If strValue = "" then
"Ce code s'exécute lorsque pifQueryNext retourne comme valeur 0.

pifLogWarningMsg("pifQueryGetStringVal returned no data. Logical.name will
be " & [MACAddress])
RetVal = [MACAddress]
Else
RetVal = strValue
End If
```

II Référence alphabétique

7 Référence alphabétique

Abs()

Syntaxe Basic interne

Function Abs(dValue As Double) As Double

Description

Renvoie la valeur absolue d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la valeur absolue.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iSeed as Integer
iSeed = Int((10*Rnd)-5)
RetVal = Abs(iSeed)
```

AppendOperand()

Syntaxe Basic interne

Function AppendOperand(strExpr As String, strOperator As String, strOperand As String) As String

Description

Concatène une chaîne en fonction des paramètres passés à la fonction. Le résultat a la forme suivante :

```
strExpr strOperator strOperand
```

Entrée

- **strExpr** : Expression à concaténer.
- **strOperator** : Opérateur à concaténer.
- **strOperand** : Opérande à concaténer.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

Si l'un des paramètres **strExpr** ou **strOperand** est omis, **strOperator** n'est pas utilisé dans la concaténation.

ApplyNewVals()

Syntaxe Basic interne

Function ApplyNewVals(strValues As String, strNewVals As String, strRows As String, strRowFormat As String) As String

Description

Affecte des valeurs identiques pour les cellules identifiées d'un contrôle "ListBox".

Entrée

- **strValues** : Chaîne source contenant les valeurs d'un contrôle "ListBox" à traiter.
- **strNewVals** : Nouvelle valeur à affecter aux cellules concernées.
- **strRows** : Identifiants des lignes à traiter. Les identifiants sont séparés par une virgule.
- **strRowFormat** : Instructions de formatage de la sous-liste. Les instructions sont séparées par le caractère "|". Chaque instruction représente le numéro de la colonne qui contiendra **strNewVals**.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Asc()

Syntaxe Basic interne

Function Asc(strAsc As String) As Long

Description

Renvoie le code ASCII du premier caractère d'une chaîne.

Entrée

- **strAsc** : Chaîne de caractères sur laquelle opère la fonction.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iCount as Integer
Dim strString as String
For iCount=Asc("A") To Asc("Z")
strString = strString & Str(iCount)
Next iCount
RetVal=strString
```

Atn()

Syntaxe Basic interne

Function Atn(dValue As Double) As Double

Description

Renvoie l'arc tangente d'un nombre, exprimé en radians

Entrée

- **dValue** : Nombre dont vous souhaitez connaître l'arc tangente.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dPi as Double
Dim strString as String
dPi=4*Atn(1)
```

```
strString = Str(dPi)
RetVal=strString
```

BasicToLocalDate()

Syntaxe Basic interne

Function BasicToLocalDate(strDateBasic As String) As String

Description

Cette fonction convertit une date au format Basic en une date au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows).

Entrée

- **strDateBasic** : Date au format Basic à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

BasicToLocalTime()

Syntaxe Basic interne

Function BasicToLocalTime(strTimeBasic As String) As String

Description

Cette fonction convertit une heure au format Basic en une heure au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows).

Entrée

- **strTimeBasic** : Heure au format Basic à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

BasicToLocalTimeStamp()

Syntaxe Basic interne

Function BasicToLocalTimeStamp(strTSBasic As String) As String

Description

Cette fonction convertit un ensemble Date+Heure au format Basic en un ensemble Date+Heure au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows).

Entrée

- **strTSBasic** : Date+Heure au format Basic à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Beep()

Syntaxe Basic interne

Function Beep()

Description

Emet un son (un beep) sur la machine.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

CDbl()

Syntaxe Basic interne

Function CDbl(dValue As Double) As Double

Description

Convertit une expression en un double ("Double").

Entrée

- **dValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dNumber As Double
Dim iInteger as Integer
iInteger = 25
dNumber=CDbl(iInteger)
RetVal=dNumber
```

ChDir()

Syntaxe Basic interne

Function ChDir(strDirectory As String)

Description

Change le répertoire courant.

Entrée

- **strDirectory** : Nouveau répertoire courant.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

ChDrive()

Syntaxe Basic interne

Function ChDrive(strDrive As String)

Description

Change le lecteur courant.

Entrée

- **strDrive** : Nouveau lecteur.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Chr()

Syntaxe Basic interne

Function Chr(lChr As Long) As String

Description

Renvoie la chaîne correspondant au code ASCII passé par le paramètre **iChr**.

Entrée

- **lChr** : Code ASCII du caractère.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iCount as Integer
Dim iIteration as Integer
Dim strMessage as String
Dim strLF as String
strLF=Chr(10)
For iIteration=1 To 2
For iCount=Asc("A") To Asc("Z")
strMessage=strMessage+Chr(iCount)
Next iCount
strMessage=strMessage+strLF
Next iIteration
RetVal=strMessage
```

CInt()

Syntaxe Basic interne

Function CInt(iValue As Long) As Long

Description

Convertit une expression en un entier ("Integer").

Entrée

- **iValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iNumber As Integer
Dim dDouble as Double
dDouble = 25.24589
iNumber=CInt(dDouble)
RetVal=iNumber
```

CLng()

Syntaxe Basic interne

Function CLng(lValue As Long) As Long

Description

Convertit une expression en un long ("Long").

Entrée

- **lValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim lNumber As Long
Dim iInteger as Integer
iInteger = 25
lNumber=CLng(iInteger)
RetVal=lNumber
```

Cos()

Syntaxe Basic interne

Function Cos(dValue As Double) As Double

Description

Renvoie le cosinus d'un nombre, exprimé en radians.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître le cosinus.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dCalc as Double
dCalc=Cos(2.79)
RetVal=dCalc
```

Remarques

Note :

La formule de conversion des degrés en radians est la suivante :

```
angle en radians = (angle en degrés) * Pi / 180
```

CountOccurences()

Syntaxe Basic interne

Function CountOccurences(strSearched As String, strPattern As String, strEscChar As String) As Long

Description

Compte le nombre d'occurrences d'une chaîne de caractères à l'intérieur d'une autre chaîne.

Entrée

- **strSearched** : Chaîne de caractères à l'intérieur de laquelle s'effectue la recherche.
- **strPattern** : Chaîne de caractères à rechercher à l'intérieur de **strSearched**.
- **strEscChar** : Caractère d'échappement. Si la fonction rencontre ce caractère à l'intérieur de la chaîne **strSearched**, la recherche s'arrête.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=CountOccurences("toi|moi|toi,moi|toi", "toi", ",") :'Renvoie la vale
ur "2"
MyStr=CountOccurences("toi|moi|toi,moi|toi", "toi", "|") :'Renvoie la vale
ur "1"
```

CountValues()

Syntaxe Basic interne

Function CountValues(strSearched As String, strSeparator As String, strEscChar As String) As Long

Description

Compte le nombre d'éléments dans une chaîne de caractères en tenant compte d'un séparateur et d'un caractère d'échappement.

Entrée

- **strSearched** : Chaîne de caractères à traiter.
- **strSeparator** : Séparateur utilisé pour délimiter les éléments.
- **strEscChar** : Caractère d'échappement. Si ce caractère préfixe un séparateur, ce dernier sera ignoré.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=CountValues("toi|moi|toi\|moi|toi", "|", "\") :'Renvoie la valeur 4
MyStr=CountValues("toi|moi|toi\|moi|toi", "|", "") :'Renvoie la valeur 5
```

CSng()

Syntaxe Basic interne

Function CSng(fValue As Single) As Single

Description

Convertit une expression en un nombre à virgule flottante ("Float").

Entrée

- **fValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dNumber As Double
Dim iInteger as Integer
iInteger = 25
dNumber=CSng(iInteger)
RetVal=dNumber
```

CStr()

Syntaxe Basic interne

Function CStr(strValue As String) As String

Description

Convertit une expression en une chaîne de caractères ("String").

Entrée

- **strValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dNumber As Double
Dim strMessage as String
dNumber = 2,452873
strMessage=CStr(dNumber)
RetVal=strMessage
```

CurDir()

Syntaxe Basic interne

Function CurDir() As String

Description

Renvoie le chemin courant.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

CVar()

Syntaxe Basic interne

Function CVar(vValue As Variant) As Variant

Description

Convertit une expression en un variant ("Variant").

Entrée

- **vValue** : Expression à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Date()

Syntaxe Basic interne

Function Date() As Date

Description

Renvoie la date courante du système.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

DateAdd()

Syntaxe Basic interne

Function DateAdd(tmStart As Date, tsDuration As Long) As Date

Description

Cette fonction calcule une nouvelle date en fonction d'une date de départ à laquelle est ajoutée une durée réelle.

Entrée

- **tmStart** : Ce paramètre contient la date à laquelle sera ajoutée une durée.
- **tsDuration** : Ce paramètre contient la durée (exprimée en secondes) à ajouter à la date **tmStart**.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

DateAddLogical()

Syntaxe Basic interne

Function DateAddLogical(tmStart As Date, tsDuration As Long) As Date

Description

Cette fonction calcule une nouvelle date en fonction d'une date de départ à laquelle est ajoutée une durée logique (un mois comporte 30 jours).

Entrée

- **tmStart** : Ce paramètre contient la date à laquelle sera ajoutée une durée.
- **tsDuration** : Ce paramètre contient la durée, exprimée en secondes, à ajouter à la date **tmStart**.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

DateDiff()

Syntaxe Basic interne

Function DateDiff(tmEnd As Date, tmStart As Date) As Date

Description

Cette fonction calcule en secondes la durée écoulée entre deux dates.

Entrée

- **tmEnd** : Ce paramètre contient la date de fin de la période sur laquelle est effectué le calcul.
- **tmStart** : Ce paramètre contient la date de début de la période sur laquelle est effectué le calcul.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant calcule la durée écoulée entre le 01/01/98 et le 01/01/99.

```
DateDiff("1998/01/01 00:00:00", "1999/01/01 00:00:00")
```

DateSerial()

Syntaxe Basic interne

Function DateSerial(iYear As Long, iMonth As Long, iDay As Long) As Date

Description

Cette fonction renvoie une date formatée en fonction des paramètres **iYear**, **iMonth** et **iDay**.

Entrée

- **iYear** : Année. Si sa valeur est comprise entre 0 et 99, ce paramètre décrit les années de 1900 à 1999. Pour toutes les autres années, vous devez utiliser un nombre de quatre chiffres (par exemple 1800).
- **iMonth** : Mois.
- **iDay** : Jour.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Chacun de ces paramètres peut prendre pour valeur une expression numérique représentant un nombre de jours, de mois ou d'années. Ainsi l'exemple suivant :

```
DateSerial(1999-10, 3-2, 15-8)
```

renvoie la valeur :

```
1989/1/7
```

Lorsque la valeur d'un paramètre est en dehors de l'intervalle de valeurs généralement admis (c'est à dire 1-31 pour les jours, 1-12 pour les mois, ...), la fonction renvoie une date vide.

DateValue()

Syntaxe Basic interne

Function DateValue(tmDate As Date) As Date

Description

Cette fonction renvoie la partie date d'une valeur "Date+Heure"

Entrée

- **tmDate** : Date au format "Date+Heure".

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant :

```
DateValue ("1999/09/24 15:00:00")
```

renvoie la valeur :

```
1999/09/24
```

Day()

Syntaxe Basic interne

Function Day(tmDate As Date) As Long

Description

Renvoie le jour contenu dans le paramètre **tmDate**.

Entrée

- **tmDate** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strDay as String
strDay=Day(Date())
RetVal=strDay
```

EscapeSeparators()

Syntaxe Basic interne

Function EscapeSeparators(strSource As String, strSeparators As String, strEscChar As String) As String

Description

Préfixe un ou plusieurs caractère(s) défini(s) comme séparateur(s) par un caractère d'échappement.

Entrée

- **strSource** : Chaîne de caractères à traiter.
- **strSeparators** : Liste des séparateurs à préfixer. Si vous souhaitez déclarer plusieurs séparateurs, vous devez les séparer par le caractère utilisé comme caractère d'échappement (indiqué dans le paramètre **strEscChar**.
- **strEscChar** : Caractère d'échappement. Il préfixera tous les séparateurs définis dans **strSeparators**.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=EscapeSeparators("toi|moi|toi,moi|toi", "|\,", "\") :'Renvoie la val
eur "toi\|moi\|toi\,moi\|toi"
```

ExeDir()

Syntaxe Basic interne

Function ExeDir() As String

Description

Cette fonction renvoie le chemin complet de l'exécutable.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strPath as string
strPath=ExeDir()
```

Exp()

Syntaxe Basic interne

Function Exp(dValue As Double) As Double

Description

Renvoie l'exponentielle d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître l'exponentielle.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iSeed as Integer
iSeed = Int((10*Rnd)-5)
RetVal = Exp(iSeed)
```

ExtractValue()

Syntaxe Basic interne

Function ExtractValue(pstrData As String, strSeparator As String, strEscChar As String) As String

Description

Extrait d'une chaîne de caractères les valeurs délimitées par un séparateur. La valeur récupérée est alors effacée de la chaîne source. Cette opération tient compte d'un éventuel caractère d'échappement. Si le séparateur n'est pas trouvé dans la chaîne source, l'intégralité de la chaîne est renvoyée et la chaîne source est entièrement effacée.

Entrée

- **pstrData** : Chaîne source à traiter.
- **strSeparator** : Caractère utilisé comme séparateur dans la chaîne source.
- **strEscChar** : Caractère d'échappement. Si ce caractère préfixe le séparateur, ce dernier est ignoré.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=ExtractValue("toi,moi", ",", "\") :'Renvoie "toi" et laisse "moi" da
ns la chaîne source
MyStr=ExtractValue(",toi,moi", ",", "\") :'Renvoie "" et laisse "toi,moi"
dans la chaîne source
MyStr=ExtractValue("toi", ",", "\") :'Renvoie "toi" et laisse "" dans la c
haîne source
MyStr=ExtractValue("toi\,moi", ",", "\") :'Renvoie "toi\,moi" et laisse ""
dans la chaîne source
MyStr=ExtractValue("toi\,moi", ",", "") :'Renvoie "toi\" et laisse "moi" d
ans la chaîne source
RetVal=""
```

FileCopy()

Syntaxe Basic interne

Function FileCopy(strSource As String, strDest As String) As Long

Description

Copie un fichier ou un répertoire.

Entrée

- **strSource** : Chemin complet du fichier ou du répertoire à copier.
- **strDest** : chemin complet du fichier ou du répertoire de destination.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

FileDateTime()

Syntaxe Basic interne

Function FileDateTime(strFileName As String) As Date

Description

Renvoie la date et l'heure d'un fichier sous la forme d'un "Long".

Entrée

- **strFileName** : Chemin complet du fichier concerné par l'opération.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

FileExists()

Syntaxe Basic interne

Function FileExists(strFileName As String) As Long

Description

Cette fonction teste l'existence d'un fichier. La fonction renvoie les valeurs suivantes :

- 0 : le fichier n'existe pas.
- 1 : le fichier existe.

Entrée

- **strFileName** : Ce paramètre contient le chemin complet du fichier dont vous souhaitez tester l'existence.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
If FileExists("c:\tmp\myfile.log") Then
strFileName = "c:\archive\"  +  FormatDate(Date, "dddd d mmm yyyy") + ".lo
g"
FileCopy("c:\tmp\myfile.log", strFileName)
End if
```

FileLen()

Syntaxe Basic interne

Function FileLen(strFileName As String) As Long

Description

Renvoie la taille d'un fichier.

Entrée

- **strFileName** : Chemin complet du fichier concerné par l'opération.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Fix()

Syntaxe Basic interne

Function Fix(dValue As Double) As Long

Description

Renvoie la partie entière (premier entier supérieur dans le cas d'un nombre négatif) d'un nombre à virgule.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la partie entière.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dSeed as Double
dSeed = (10*Rnd)-5
RetVal = Fix(dSeed)
```

FormatDate()

Syntaxe Basic interne

Function FormatDate(tmFormat As Date, strFormat As String) As String

Description

Formate une date en fonction de l'expression contenue dans le paramètre **strFormat**.

Entrée

- **tmFormat** : Date à formater.
- **strFormat** : Expression contenant les instructions de formatage.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple de code suivant montre comment formater une date :

```
Dim MyDate
MyDate="2000/03/14"
RetVal=FormatDate(MyDate, "dddd d mmmm yyyy") :'Renvoie "Tuesday 14 March
2000"
```

FormatResString()

Syntaxe Basic interne

Function FormatResString(strResString As String, strParamOne As String, strParamTwo As String, strParamThree As String, strParamFour As String, strParamFive As String) As String

Description

Cette fonction traite une chaîne source en remplaçant les variables $1, $2, $3, $4 et $5 respectivement par les chaînes contenues dans les paramètres **strParamOne**, **strParamTwo**, **strParamThree**, **strParamFour** et **strParamFive**.

Entrée

- **strResString** : Chaîne source à traiter.
- **strParamOne** : Chaîne de remplacement de la variable $1.
- **strParamTwo** : Chaîne de remplacement de la variable $2.
- **strParamThree** : Chaîne de remplacement de la variable $3.
- **strParamFour** : Chaîne de remplacement de la variable $4.
- **strParamFive** : Chaîne de remplacement de la variable $5.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant :

```
FormatResString("je$1il$2vous$3", "tu", "nous", "ils")
```

renvoie "jetuilnousvousils".

FV()

Syntaxe Basic interne

Function FV(dblRate As Double, iNper As Long, dblPmt As Double, dblPV As Double, iType As Long) As Double

Description

Cette fonction renvoie le futur montant d'une annuité basée sur des versements constants et périodiques, et sur un taux d'intérêt fixe.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière.
- **dblPmt** : Ce paramètre indique le montant du paiement à effectuer à chaque échéance. Le paiement comporte généralement le principal et les intérêts.
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les paramètres **Rate** et **Nper** doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités.
- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

GetEnvVar()

Syntaxe Basic interne

Function GetEnvVar(strVar As String, bExpand As Long) As String

Description

Cette fonction renvoie la valeur d'une variable d'environnement. Une valeur vide est renvoyée si la variable d'environnement n'existe pas.

Entrée

- **strVar** : Ce paramètre contient le nom de la variable d'environnement.
- **bExpand** : Ce paramètre booléen est utile quand la variable d'environnement fait référence à une ou plusieurs autres variables d'environnement. Dans ce cas, quand ce paramètre a pour valeur 1 (valeur par défaut), chaque variable référencée est remplacée par sa valeur. Dans le cas contraire, elle ne l'est pas.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
RetVal = getEnvVar("PROMPT")
```

GetListItem()

Syntaxe Basic interne

Function GetListItem(strFrom As String, strSep As String, lNb As Long, strEscChar As String) As String

Description

Renvoie la **lNb**ième portion d'une chaîne délimitée par des séparateurs.

Entrée

- **strFrom** : Chaîne source à traiter.
- **strSep** : Caractère utilisé comme séparateur dans la chaîne source.
- **lNb** : Position de la chaîne à récupérer.

- **strEscChar** : Caractère d'échappement. Si ce caractère préfixe un séparateur, ce dernier sera ignoré.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant :

```
GetListItem("ceci_est_un_test", "_", 2, "%")
```

renvoie "est".

```
GetListItem("ceci%_est_un_test", "_", 2, "%")
```

renvoie "un".

GetXmlAttributeValue()

Syntaxe Basic interne

Function GetXmlAttributeValue(strVal As String) As String

Description

Cette fonction permet d'échapper les caractères non valides pour une valeur d'élément XML donnée.

Entrée

Ce paramètre contient le caractère à échapper.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Par exemple, la fonction suivante transforme le caractère simple guillemet en caractère valide :

```
GetXmlAttributeValue("doesn't")
```

Le résultat de la transformation est le suivant :

```
"doesn&quote;t"
```

GetXmlElementValue()

Syntaxe Basic interne

Function GetXmlElementValue(strElemContent As String) As String

Description

Cette fonction permet d'échapper les caractères non valides pour une valeur d'attribut XML donnée.

Entrée

Ce paramètre contient le caractère à échapper.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Vous pouvez utiliser cette fonction dans une clause WHERE comme décrit ci-desssous :

```
strLayer = strLayer + "<Where Path='ItemsUsed'>" + GetXmlElementValue("Ass
etTag like 'A%'") + "</Where>"
```

Voir aussi l'exemple commenté pour la fonction : PifNewQueryFromFmtName() [page 153]

Hex()

Syntaxe Basic interne

Function Hex(dValue As Double) As String

Description

Renvoie la valeur hexadécimale d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la valeur hexadécimale.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Hour()

Syntaxe Basic interne

Function Hour(tmTime As Date) As Long

Description

Renvoie la valeur de l'heure contenue dans le paramètre **tmTime**.

Entrée

- **tmTime** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strHour as String
strHour=Hour(Date())
RetVal=strHour
```

InStr()

Syntaxe Basic interne

Function InStr(lPosition As Long, strSource As String, strPattern As String, bCaseSensitive As Long) As Long

Description

Renvoie la position de la première occurrence d'une chaîne de caractères à l'intérieur d'une autre chaîne de caractères.

Entrée

- **lPosition** : Position de départ de la recherche. Ce paramètre ne peut être négatif et ne doit pas dépasser 65.535.
- **strSource** : Chaîne dans laquelle s'effectue la recherche.
- **strPattern** : Chaîne de caractères à rechercher.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strSource as String
Dim strToSearch as String
Dim iPosition
strSource = "Good Bye"
strToSearch = "Bye"
iPosition = Instr(2, strSource, strToSearch)
RetVal=iPosition
```

Remarques

Note :

La position de la première occurence est toujours 1. La fonction renvoie 0 si la chaîne de caractères à rechercher n'est pas trouvée.

Int()

Syntaxe Basic interne

Function Int(dValue As Double) As Long

Description

Renvoie la partie entière (premier nombre entier inférieur dans le cas d'un nombre négatif) d'un nombre à virgule.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la partie entière.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iSeed as Integer
iSeed = Int((10*Rnd)-5)
RetVal = Abs(iSeed)
```

IPMT()

Syntaxe Basic interne

Function IPMT(dblRate As Double, iPer As Long, iNper As Long, dblPV As Double, dblFV As Double, iType As Long) As Double

Description

Cette fonction renvoie le montant des intérêts pour une échéance donnée d'une annuité.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **iPer** : Ce paramètre indique la période concernée par le calcul, comprise entre 1 et la valeur du paramètre **Nper**.
- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière.
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.
- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les paramètres **Rate** et **Nper** doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités.
- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

IsNumeric()

Syntaxe Basic interne

Function IsNumeric(strString As String) As Long

Description

Cette fonction permet de déterminer si une chaîne de caractères est un nombre.

Entrée

- **strString** : ce paramètre contient la chaîne de caractères à évaluer.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Kill()

Syntaxe Basic interne

Function Kill(strKilledFile As String) As Long

Description

Efface un fichier.

Entrée

- **strKilledFile** : Chemin complet du fichier concerné par l'opération.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

LCase()

Syntaxe Basic interne

Function LCase(strString As String) As String

Description

Passe tous les caractères d'une chaîne en minuscules.

Entrée

- **strString** : Chaîne de caractères à passer en minuscules.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
' This example uses the LTrim and RTrim functions to strip leading ' and t
railing spaces, respectively, from a string variable.
' It uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Dim strString as String
Dim strTrimString as String
strString = "  <-Trim->  " :' Initialize string.
strTrimString = LTrim(strString) :' strTrimString = "<-Trim->  ".
strTrimString = LCase(RTrim(strString)) :' strTrimString = "  <-trim->".
strTrimString = LTrim(RTrim(strString)) :' strTrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
```

```
strTrimString = UCase(Trim(strString)) :' strTrimString = "<-TRIM->".
RetVal= "|" & strTrimString & "|"
```

Left()

Syntaxe Basic interne

Function Left(strString As String, lNumber As Long) As String

Description

Renvoie les iNumber premiers caractères d'une chaîne en partant de la gauche.

Entrée

- **strString** : Chaîne de caractères à traiter.
- **lNumber** : Nombre de caractères à renvoyer.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim lWord, strMsg, rWord, iPos :' Declare variables.
strMsg = "Left() Test."
iPos = InStr(1, strMsg, " ") :' Find space.
lWord = Left(strMsg, iPos - 1) :' Get left word.
rWord = Right(strMsg, Len(strMsg) - iPos) :' Get right word.
strMsg=rWord+lWord :' And swap them
RetVal=strMsg
```

LeftPart()

Syntaxe Basic interne

Function LeftPart(strFrom As String, strSep As String, bCaseSensitive As Long) As String

Description

Extrait la portion d'une chaîne de caractères située à gauche du séparateur précisé dans le paramètre **strSep**.

La recherche du séparateur s'effectue de la gauche vers la droite.

La recherche peut tenir compte ou non de la casse en fonction de la valeur du paramètre **bCaseSensitive**.

Entrée

- **strFrom** : Chaîne source à traiter.
- **strSep** : Caractère utilisé comme séparateur dans la chaîne source.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Ces exemples illustrent l'utilisation des fonctions **LeftPart**, **LeftPartFromRight**, **RightPart** et **RightPartFromLeft** sur une même chaîne de caractères: "Ceci_est_un_test" :

```
LeftPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci".

```
LeftPartFromRight("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci_est_un".

```
RightPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "test".

```
RightPartFromLeft("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "est_un_test".

LeftPartFromRight()

Syntaxe Basic interne

Function LeftPartFromRight(strFrom As String, strSep As String, bCaseSensitive As Long) As String

Description

Extrait la portion d'une chaîne de caractères située à gauche du séparateur précisé dans le paramètre **strSep**.

La recherche du séparateur s'effectue de la droite vers la gauche.

La recherche peut tenir compte ou non de la casse en fonction de la valeur du paramètre **bCaseSensitive**.

Entrée

- **strFrom** : Chaîne source à traiter.
- **strSep** : Caractère utilisé comme séparateur dans la chaîne source.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Ces exemples illustrent l'utilisation des fonctions **LeftPart**, **LeftPartFromRight**, **RightPart** et **RightPartFromLeft** sur une même chaîne de caractères: "Ceci_est_un_test" :

```
LeftPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci".

```
LeftPartFromRight("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci_est_un".

```
RightPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "test".

```
RightPartFromLeft("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "est_un_test".

Len()

Syntaxe Basic interne

Function Len(vValue As Variant) As Long

Description

Renvoie le nombre de caractères d'une chaîne ou d'un variant.

Entrée

- **vValue** : Variant concerné par l'opération.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strTest as String
Dim iLength as Integer
strTest = "Peregrine Systems"
iLength = Len(strTest) :'The value of iLength is 17
RetVal=iLength
```

LocalToBasicDate()

Syntaxe Basic interne

Function LocalToBasicDate(strDateLocal As String) As String

Description

Cette fonction convertit une date au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows) en une date au format Basic.

Entrée

- **strDateLocal** : Date au format chaîne à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

LocalToBasicTime()

Syntaxe Basic interne

Function LocalToBasicTime(strTimeLocal As String) As String

Description

Cette fonction convertit une heure au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows) en une heure au format Basic.

Entrée

- **strTimeLocal** : Heure au format chaîne à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

LocalToBasicTimeStamp()

Syntaxe Basic interne

Function LocalToBasicTimeStamp(strTSLocal As String) As String

Description

Cette fonction convertit un ensemble Date+Heure au format chaîne (telle qu'elle est affichée dans le "control panel" de Windows) en un ensemble Date+Heure au format Basic.

Entrée

- **strTSLocal** : Date+Heure au format chaîne à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

LocalToUTCDate()

Syntaxe Basic interne

Function LocalToUTCDate(tmLocal As Date) As Date

Description

Cette fonction convertit une date au format "Date+Heure" en une date au format UTC (indépendante d'un quelconque fuseau horaire).

Entrée

- **tmLocal** : Date au format "Date+Heure".

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Log()

Syntaxe Basic interne

Function Log(dValue As Double) As Double

Description

Renvoie le logarithme néperien d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître le logarithme.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dSeed as Double
dSeed = Int((10*Rnd)-5)
RetVal = Log(dSeed)
```

LTrim()

Syntaxe Basic interne

Function LTrim(strString As String) As String

Description

Supprime tous les espaces précédant le premier caractère (différent d'un espace) d'une chaîne.

Entrée

- **strString** : Chaîne de caractères à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
' This example uses the LTrim and RTrim functions to strip leading ' and t
railing spaces, respectively, from a string variable.
' It uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Dim strString as String
Dim strTrimString as String
strString = "  <-Trim->  " :' Initialize string.
strTrimString = LTrim(strString) :' strTrimString = "<-Trim->  ".
strTrimString = LCase(RTrim(strString)) :' strTrimString = "  <-trim->".
strTrimString = LTrim(RTrim(strString)) :' strTrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
strTrimString = UCase(Trim(strString)) :' strTrimString = "<-TRIM->".
RetVal= "|" & strTrimString & "|"
```

MakeInvertBool()

Syntaxe Basic interne

Function MakeInvertBool(lValue As Long) As Long

Description

Cette fonction renvoie un booléen inversé (0 devient 1, tout autre nombre devient 0).

Entrée

- **lValue** : Nombre concerné par l'opération.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyValue
MyValue=MakeInvertBool(0) :'Renvoie la valeur 1
MyValue=MakeInvertBool(1) :'Renvoie la valeur 0
MyValue=MakeInvertBool(254) :'Renvoie la valeur 0
```

Mid()

Syntaxe Basic interne

Function Mid(strString As String, lStart As Long, lLen As Long) As String

Description

Extrait une chaîne de caractères contenue dans une autre chaîne.

Entrée

- **strString** : Chaîne de caractères concernée par l'opération.
- **lStart** : Position de départ de la chaîne à extraire à l'intérieur de strString.
- **lLen** : longueur de la chaîne à extraire.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strTest as String
strTest="One Two Three" :' Defines the test string
strTest=Mid(strTest,5,3) :' strTest="Two"
RetVal=strTest
```

Minute()

Syntaxe Basic interne

Function Minute(tmTime As Date) As Long

Description

Renvoie le nombre de minutes contenues l'heure exprimée par le paramètre **tmTime**.

Entrée

- **tmTime** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strMinute
strMinute=Minute(Date())
RetVal=strMinute :'Renvoie le nombre de minutes écoulées dans l'heure cour
ante par exemple "45" s'il est actuellement 15:45:30
```

MkDir()

Syntaxe Basic interne

Function MkDir(strMkDirectory As String) As Long

Description

Crée un répertoire.

Entrée

- **strMkDirectory** : Chemin complet du répertoire à créer.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim lErr as Long
' Create the c:\tmp directory
lErr = MkDir("c:\tmp")
```

Month()

Syntaxe Basic interne

Function Month(tmDate As Date) As Long

Description

Renvoie le mois contenu dans la date exprimée par le paramètre **tmDate**.

Entrée

- **tmDate** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim lMonth as Long
lMonth=Month(Date())
RetVal=lMonth :'Renvoie le mois courant
```

Name()

Syntaxe Basic interne

Function Name(strSource As String, strDest As String)

Description

Renomme un fichier.

Entrée

- **strSource** : Chemin complet du fichier à renommer.
- **strDest** : Nouveau nom du fichier.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim lErr as Long
' Rename "C:\tmp\src.txt" as "D:\tmp\dst.txt"
lErr = Name("C:\tmp\src.txt", "D:\tmp\dst.txt")
```

Now()

Syntaxe Basic interne

Function Now() As Date

Description

Renvoie la date et l'heure courantes.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

NPER()

Syntaxe Basic interne

Function NPER(dblRate As Double, dblPmt As Double, dblPV As Double, dblFV As Double, iType As Long) As Double

Description

Cette fonction renvoie le nombre d'échéances d'une annuité basée sur des versements constants et périodiques, et sur un taux d'intérêt fixe.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **dblPmt** : Ce paramètre indique le montant du paiement à effectuer à chaque échéance. Le paiement comporte généralement le principal et les intérêts.
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.
- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

Oct()

Syntaxe Basic interne

Function Oct(dValue As Double) As String

Description

Renvoie la valeur octale d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la valeur octale.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dSeed as Double
dSeed = Int((10*Rnd)-5)
RetVal = Oct(dSeed)
```

ParseDate()

Syntaxe Basic interne

Function ParseDate(strDate As String, strFormat As String, strStep As String) As Date

Description

Cette fonction convertit une date exprimée sous la forme d'une chaîne de caractères en un objet date au sens Basic du terme.

Entrée

- **strDate** : Date au format chaîne de caractères.
- **strFormat** : Ce paramètre contient le format de la date contenue dans la chaîne de caractères. Les valeurs possibles sont les suivantes :
  - jj/mm/aa
  - jj/mm/aaaa
  - mm/jj/aa
  - mm/jj/aaaa
  - aaaa/mm/jj
  - Date : date exprimée suivant les paramètres de date du poste client.
  - DateInter : date exprimée au format international
- **strStep** : Ce paramètre optionnel contient le séparateur de date utilisé dans la chaîne de caractères. Les séparateurs autorisés sont "\" et "-".

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dDate as date
dDate=ParseDate("2001/05/01", "aaaa/mm/jj")
```

ParseDMYDate()

Syntaxe Basic interne

Function ParseDMYDate(strDate As String) As Date

Description

Cette fonction renvoie un objet Date (au sens Basic) à partir d'une date formatée comme suit :

```
jj/mm/aaaa
```

Entrée

- **strDate** : Date stockée sous la forme d'une chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dDate as Date
dDate = ParseDMYDate("31/02/2003")
```

ParseMDYDate()

Syntaxe Basic interne

Function ParseMDYDate(strDate As String) As Date

Description

Cette fonction renvoie un objet Date (au sens Basic) à partir d'une date formatée comme suit :

```
mm/jj/aaaa
```

Entrée

- **strDate** : Date stockée sous la forme d'une chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dDate as Date
dDate = ParseMDYDate("02/31/2003")
```

ParseYMDDate()

Syntaxe Basic interne

Function ParseYMDDate(strDate As String) As Date

Description

Cette fonction renvoie un objet Date (au sens Basic) à partir d'une date formatée comme suit :

```
aaaa/mm/jj
```

Entrée

- **strDate** : Date stockée sous la forme d'une chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dDate as Date
dDate = ParseYMDDate("2003/02/31")
```

PifCloseODBCDatabase()

Syntaxe Basic interne

Function PifCloseODBCDatabase(strDSN As String) As Long

Description

Cette fonction met fin à une connexion ODBC.

Entrée

- **strDSN** : Nom de la source de données de la connexion ODBC.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim iRet As Integer
iRet = PifOpenOdbcDatabase(PifStrVal("DSN"), PifStrVal("LOGIN"), PifStrVal
("PWD"))

if  iRet = 0 Then
Dim strQuery As String
strQuery = "SELECT a.AssetName, a.FullName FROM Asset a"
iRet = PifCreateDynaMapTable(PifStrVal("DSN"), "Assets", strQuery, FALSE)
End If

Dim strRes as String
if  iRet = 0 Then
strRes = PifMapValue([FullName], "Assets", 1, "", 1)
End If

If iRet = 0 and strRes = "" Then
iRet = PifExecODBCSql(PifStrVal("DSN"), "INSERT INTO Asset(AssetNAme, Asse
tFullName)
```

PifCreateDynaMaptableFromFmtName()

Syntaxe Basic interne

Function PifCreateDynaMaptableFromFmtName(strCntrName As String, strFmtName As String, strLayer As String, strMaptableName As String, strKeyName As String, bCreateOnce As Long) As Long

Description

Cette fonction permet de créer de manière dynamique une table de correspondance au démarrage d'une session.

Entrée

- **strCntrName** : Nom du connecteur.
- **strFmtName** : Type de document à produire qui va définir le contenu de la table de correspondance.
- **strLayer** : Clause Where appliquée au type de document.
- **strMaptableName** : Nom de la table de correspondance à créer.
- **strKeyName** : Nom de la colonne utilisé comme clé pour la table de correspondance à créer.

  Si aucun nom de colonne n'est précisé comme clé, le premier attribut de la requête ayant servi à créer la table de correspondance est utilisé.

  Par exemple, pour utiliser l'identifiant d'un élément de parc, la syntaxe sera la suivante :

  ```
  Portfolio.AssetTag
  ```

  Il n'est pas nécessaire de préciser le nom du type de document dans cette syntaxe, la table de correspondance s'appliquant au type de document courant.
- **bCreateOnce** :
  - **0** : la table de correspondance est créée à chaque fois que la fonction est appelée.
  - **1** : la table de correspondance est créée pour toute la durée de la session.

Sortie

- 0 : La fonction s'est exécutée normalement.

- Non nul : Code d'erreur.

Remarques

Voir aussi :

- PifMapValueEx() [page 151]
- PifMapValueContainingEx() [page 149]

PifDateToTimezone()

Syntaxe Basic interne

Function PifDateToTimezone(tmSrc As Date, strTmznSrc As String, bWithDayLightSavingSrc As Long, strTmznDst As String, bDayLightSavingDst As Long) As Date

Description

Cette fonction convertit une date (au sens date et heure) exprimée dans un fuseau horaire donné dans un autre fuseau horaire en prenant en compte, si besoin est, les paramètres d'heure d'hiver.

Un message d'erreur est renvoyé si le fuseau horaire spécifié n'existe pas.

Note :

Le fuseau horaire 'UTC' (Universal Time Coordinated) peut être utilisé pour spécifier une date GMT 0 sans heure d'hiver.

Entrée

- **tmSrc** : Ce paramètre contient la date à convertir.
- **strTmznSrc** : Ce paramètre contient le nom du fuseau horaire source pour la conversion.
- **bWithDayLightSavingSrc** : En fonction de la valeur de ce paramètre, la fonction tient (=1) ou non (=0) compte des paramètres de l'heure d'hiver pour le fuseau horaire source.
- **strTmznDst** : Ce paramètre contient le nom du fuseau horaire de destination pour la conversion.

- **bDayLightSavingDst** : En fonction de la valeur de ce paramètre, la fonction tient (=1) ou non (=0) compte des paramètres de l'heure d'hiver pour le fuseau horaire destination.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
PifDateToTimezone("2002-08-29 12:00:00", "Romance Standard Time", 1, "UTC"
, 0)
```

renvoie la valeur :

```
2002-08-29 10:00:00
```

```
PifDateToTimezone("2002-08-29 12:00:00", "Romance Standard Time", 0, "UTC"
, 0)
```

renvoie la valeur :

```
2002-08-29 11:00:00
```

```
PifDateToTimezone("2002-12-25 12:00:00", "Romance Standard Time", 1, "UTC"
, 0)
```

renvoie la valeur :

```
2002-12-25 11:00:00
```

Remarques

Liste et description des fuseaux horaires :

Nom	Description
Dateline Standard Time	(GMT-12:00) Eniwetok, Kwajalein
Samoa Standard Time	(GMT-11:00) Midway Island, Samoa
Hawaiian Standard Time	(GMT-10:00) Hawaii
Alaskan Standard Time	(GMT-09:00) Alaska
Pacific Standard Time	(GMT-08:00) Pacific Time (US & Canada); Tijuana
US Mountain Standard Time	(GMT-07:00) Arizona
Mountain Standard Time	(GMT-07:00) Mountain Time (US & Canada)
Central America Standard Time	(GMT-06:00) Central America
Central Standard Time	(GMT-06:00) Central Time (US & Canada)
Mexico Standard Time	(GMT-06:00) Mexico City
Canada Central Standard Time	(GMT-06:00) Saskatchewan

Nom	Description
SA Pacific Standard Time	(GMT-05:00) Bogota, Lima, Quito
Eastern Standard Time	(GMT-05:00) Eastern Time (US & Canada)
US Eastern Standard Time	(GMT-05:00) Indiana (East)
Atlantic Standard Time	(GMT-04:00) Atlantic Time (Canada)
SA Western Standard Time	(GMT-04:00) Caracas, La Paz
Pacific SA Standard Time	(GMT-04:00) Santiago
Newfoundland Standard Time	(GMT-03:30) Newfoundland
E. South America Standard Time	(GMT-03:00) Brasilia
SA Eastern Standard Time	(GMT-03:00) Buenos Aires, Georgetown
Greenland Standard Time	(GMT-03:00) Greenland
Mid-Atlantic Standard Time	(GMT-02:00) Mid-Atlantic
Azores Standard Time	(GMT-01:00) Azores
Cape Verde Standard Time	(GMT-01:00) Cape Verde Is.
Greenwich Standard Time	(GMT) Casablanca, Monrovia
GMT Standard Time	(GMT) Greenwich Mean Time : Dublin, Edinburgh, Lisbon, London
UTC	(GMT) Universal Coordinated Time
W. Europe Standard Time	(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Central Europe Standard Time	(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
Romance Standard Time	(GMT+01:00) Brussels, Copenhagen, Madrid, Paris
Central European Standard Time	(GMT+01:00) Sarajevo, Skopje, Sofija, Vilnius, Warsaw, Zagreb
W. Central Africa Standard Time	(GMT+01:00) West Central Africa
GTB Standard Time	(GMT+02:00) Athens, Istanbul, Minsk
E. Europe Standard Time	(GMT+02:00) Bucharest
Egypt Standard Time	(GMT+02:00) Cairo
South Africa Standard Time	(GMT+02:00) Harare, Pretoria
FLE Standard Time	(GMT+02:00) Helsinki, Riga, Tallinn
Jerusalem Standard Time	(GMT+02:00) Jerusalem
Arabic Standard Time	(GMT+03:00) Baghdad
Arab Standard Time	(GMT+03:00) Kuwait, Riyadh
Russian Standard Time	(GMT+03:00) Moscow, St. Petersburg, Volgograd
E. Africa Standard Time	(GMT+03:00) Nairobi
Iran Standard Time	(GMT+03:30) Tehran
Arabian Standard Time	(GMT+04:00) Abu Dhabi, Muscat
Caucasus Standard Time	(GMT+04:00) Baku, Tbilisi, Yerevan
Afghanistan Standard Time	(GMT+04:30) Kabul
Ekaterinburg Standard Time	(GMT+05:00) Ekaterinburg
West Asia Standard Time	(GMT+05:00) Islamabad, Karachi, Tashkent
India Standard Time	(GMT+05:30) Calcutta, Chennai, Mumbai, New Delhi
Nepal Standard Time	(GMT+05:45) Kathmandu
N. Central Asia Standard Time	(GMT+06:00) Almaty, Novosibirsk
Central Asia Standard Time	(GMT+06:00) Astana, Dhaka

Nom	Description
Sri Lanka Standard Time	(GMT+06:00) Sri Jayawardenepura
Myanmar Standard Time	(GMT+06:30) Rangoon
SE Asia Standard Time	(GMT+07:00) Bangkok, Hanoi, Jakarta
North Asia Standard Time	(GMT+07:00) Krasnoyarsk
China Standard Time	(GMT+08:00) Beijing, Chongqing, Hong Kong, Urumqi
North Asia East Standard Time	(GMT+08:00) Irkutsk, Ulaan Bataar
Malay Peninsula Standard Time	(GMT+08:00) Kuala Lumpur, Singapore
W. Australia Standard Time	(GMT+08:00) Perth
Taipei Standard Time	(GMT+08:00) Taipei
Tokyo Standard Time	(GMT+09:00) Osaka, Sapporo, Tokyo
Korea Standard Time	(GMT+09:00) Seoul
Yakutsk Standard Time	(GMT+09:00) Yakutsk
Cen. Australia Standard Time	(GMT+09:30) Adelaide
AUS Central Standard Time	(GMT+09:30) Darwin
E. Australia Standard Time	(GMT+10:00) Brisbane
AUS Eastern Standard Time	(GMT+10:00) Canberra, Melbourne, Sydney
West Pacific Standard Time	(GMT+10:00) Guam, Port Moresby
Tasmania Standard Time	(GMT+10:00) Hobart
Vladivostok Standard Time	(GMT+10:00) Vladivostok
Central Pacific Standard Time	(GMT+11:00) Magadan, Solomon Is., New Caledonia
New Zealand Standard Time	(GMT+12:00) Auckland, Wellington
Fiji Standard Time	(GMT+12:00) Fiji, Kamchatka, Marshall Is.
Tonga Standard Time	(GMT+13:00) Nuku'alofa

PifExecODBCSql()

Syntaxe Basic interne

Function PifExecODBCSql(strDSN As String, strSqlQuery As String) As Variant

Description

Cette fonction exécute une requête SQL sur une base de données ODBC.

Entrée

- **strDSN** : Nom de la source de données de la connexion ODBC.
- **strSqlQuery** : Requête SQL à exécuter sur la base de données ODBC.

Sortie

La fonction renvoie le premier résultat de la requête. Si aucun résultat ne correspond à la requête, la fonction renvoie un NULL.

Exemple

```
Dim iRet As Integer
iRet = PifOpenOdbcDatabase(PifStrVal("DSN"), PifStrVal("LOGIN"), PifStrVal
("PWD"))

if  iRet = 0 Then
Dim strQuery As String
strQuery = "SELECT a.AssetName, a.FullName FROM Asset a"
iRet = PifCreateDynaMapTable(PifStrVal("DSN"), "Assets", strQuery, FALSE)
End If

Dim strRes as String
if  iRet = 0 Then
strRes = PifMapValue([FullName], "Assets", 1, "", 1)
End If

If iRet = 0 and strRes = "" Then
iRet = PifExecODBCSql(PifStrVal("DSN"), "INSERT INTO Asset(AssetNAme, Asse
tFullName)
```

PifFirstInCol()

Syntaxe Basic interne

Function PifFirstInCol(strPathCol As String, strChildCond As String, iStartCount As Long) As Long

Description

Cette fonction renvoie le numéro du premier élément d'une collection qui remplit la condition exprimée dans le paramètre **strChildCond**.

Entrée

- **strPathCol** : Nom (chemin) de la collection sur laquelle s'effectue la recherche.

- **strChildCond** : Condition de recherche sur l'élément. Cette condition est de la forme :

```
<Chemin> = <Valeur>
```

où :

  - <*Chemin*> est le chemin d'un élément de la collection
  - <*Valeur*> est une valeur exprimée sous la forme d'une chaîne de caractères dans la locale de Connect-It

- **iStartCount** : Numéro (index) à partir duquel la recherche commence.

> Note :
>
> Si n représente le nombre d'éléments de la collection, **iStartCount** doit être compris entre 0 et n-1.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim iToTal As Integer
Dim iIndex As Integer
iToTal = 0
iIndex = 0

Do
iIndex = PifFirstInCol("Software", "Brand=Peregrine", 0)
If iIndex <> 0 Then
iToTal = iToTal + 1
End If
Loop Until iIndex = 0
```

PifGetBlobSize()

Syntaxe Basic interne

Function PifGetBlobSize(strPath As String) As Long

Description

Cette fonction renvoie la taille d'un objet blob.

Entrée

- **strPath** : Ce paramètre contient le chemin complet de l'objet blob dans la collection.

Sortie

La fonction renvoie la taille de l'objet blob identifié par son chemin complet.

Exemple

```
Dim iSize a Integer
iSize = PifGetBlobSize("Description")
```

PifGetDateVal()

Syntaxe Basic interne

Function PifGetDateVal(strPath As String, bCkeckNodeExists As Long) As Date

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la stocke dans une variable basic de type date.

Entrée

strPath : Désigne un noeud de type de document de type date.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format de type date.

Exemple

```
Dim DateVal As Date
DateVal = PifGetDateVal("champ contenant une date")
RetVal = DateVal
```

PifGetDoubleVal()

Syntaxe Basic interne

Function PifGetDoubleVal(strPath As String, bCkeckNodeExists As Long) As Double

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie en nombre de type double.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format nombre de type double.

Exemple

```
RetVal = PifGetDoubleVal("champ contenant un nombre de type double")
```

PifGetElementChildName()

Syntaxe Basic interne

Function PifGetElementChildName(strPath As String, iItem As Long) As String

Description

Cette fonction renvoie le nom du ième sous-élément d'un noeud identifié d'un type de document source.

Entrée

- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.
- **iItem** : Ce paramètre contient le numéro du sous-élément dont vous souhaitez récupérer le nom.

Sortie

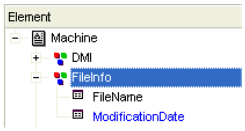
La fonction renvoie le nom de l'élément.

Note :

Une erreur est consignée dans les lignes de suivi et une chaîne vide est renvoyée par la fonction si le paramètre **iItem** est négatif ou supérieur au nombre de sous-éléments du noeud.

Exemple

Si l'on considère le type de document source suivant :




```
pifGetElementChildName("FileInfo",0)
```

renvoie "FileName"

```
pifGetElementChildName("FileInfo",1)
```

renvoie "ModificationDate"

```
pifGetElementChildName("FileInfo",2)
```

renvoie une chaîne vide et consigne une erreur dans les lignes de suivi.

PifGetElementCount()

Syntaxe Basic interne

Function PifGetElementCount(strPath As String) As Long

Description

Cette fonction renvoie le nombre de sous-éléments d'un noeud identifié d'un type de document source.

Entrée

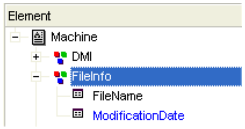
- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.

Sortie

La fonction renvoie le nombre de sous-élements du noeud dont le chemin est précisé par **strPath**.

Exemple

Si l'on considère le type de document source suivant :



Le script suivant :

```
Dim iChildCount as Integer
iChildCount = PifGetElementCount("FileInfo")
```

renvoie la valeur 2. L'élément **FileInfo** possède en effet deux sous-éléments : **FileName** et **ModificationDate**.

PifGetHexStringFromBlob()

Syntaxe Basic interne

Function PifGetHexStringFromBlob(strPath As String, bCkeckNodeExists As Long) As Long

Description

Cette fonction permet de stocker un objet binaire long (blob) du document source dans une chaîne de caractère hexadécimale. Chaque bit de l'objet binaire est stocké sous sa forme hexadécimale (sur deux caractères) dans la chaîne. Par exemple, la représentation hexadécimale de la valeur décimale "27" est "1B".

Entrée

- **strPath** : Ce paramètre contient le chemin complet de l'élément binaire (blob) dans le document source.
- **bCkeckNodeExists** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) si l'élément binaire est introuvable dans le document source. Par défaut ce paramètre a pour valeur FALSE.

Sortie

La fonction renvoie la représentation hexadécimale de l'élément binaire.

Exemple

Sur certains serveurs LDAP, une entrée est définie de façon unique par l'élément binaire "ObjectGUID". Le script ci-dessous importe cette valeur binaire dans une chaîne de caractères du connecteur destination et l'utilise comme clé de réconciliation.

```
RetVal = PifGetHexStringFromBlob("ObjectGUID", TRUE)
```

PifGetInstance()

Syntaxe Basic interne

Function PifGetInstance() As String

Description

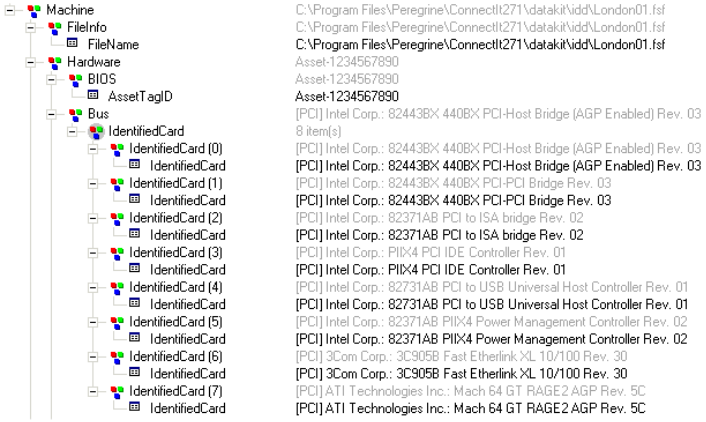
Dans un mapping collection à collection, cette fonction renvoie le numéro de l'élément actuellement traité au sein de la collection. Le premier élément traité a pour numéro "0".

Sortie

Le numéro de l'élément traité est renvoyé sous la forme d'une chaîne de caractères.

Exemple

Si l'on considère le mapping de collection suivant (sur **Hardware.Bus.IdentificationCard**) :



alors le script suivant :

```
Dim strMyElement as String
strMyElement= "Item #" & PifGetInstance
```

renvoie les valeurs **Item #0**, **Item #1**, **Item #2**, etc.

PifGetIntlStringVariantVal()

Syntaxe Basic interne

Function PifGetIntlStringVariantVal(strPath As String, bCkeckNodeExists As Long) As String

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie en chaîne internationale.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format chaîne internationale.

Exemple

```
Dim strTest As String
strTest = PifGetIntlStringVariantVal("int64")
PifLogInfoMsg(CStr([int64]))
RetVal = strTest
```

Le résultat de cette fonction est que la valeur de strTest est la valeur au format entier 64 convertie au format chaîne. La valeur affichée au journal est 281478209994880 et correspond à la valeur 2.8147820999488e+014 traitée comme un nombre de type double par une fonction Basic (CStr).

Remarques

Cette fonction est destinée à palier une limitation du moteur Basic qui ne supporte pas les entiers 64.

PifGetIntVal()

Syntaxe Basic interne

Function PifGetIntVal(strPath As String, bCkeckNodeExists As Long) As Long

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie en nombre de type entier.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format nombre de type entier.

Exemple

```
Dim IValue As Int
IValue = PifGetIntVal("champ contenant un entier")
RetVal = IValue
```

PifGetItemCount()

Syntaxe Basic interne

Function PifGetItemCount(strPath As String) As Long

Description

Cette fonction renvoie le nombre d'éléments dans une collection identifiée par son chemin d'accès.

Entrée

- **strPath** : Chemin d'accès complet de la collection concernée par l'opération.

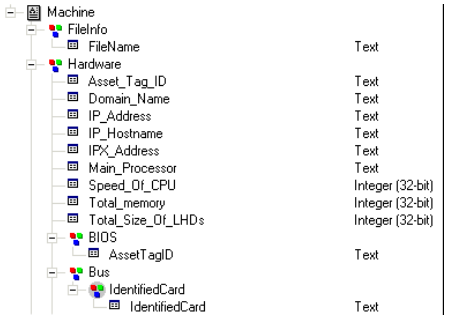
Sortie

La fonction renvoie le nombre d'éléments dans la collection.

Si la collection n'existe pas, la fonction renvoie la valeur "0".

Exemple

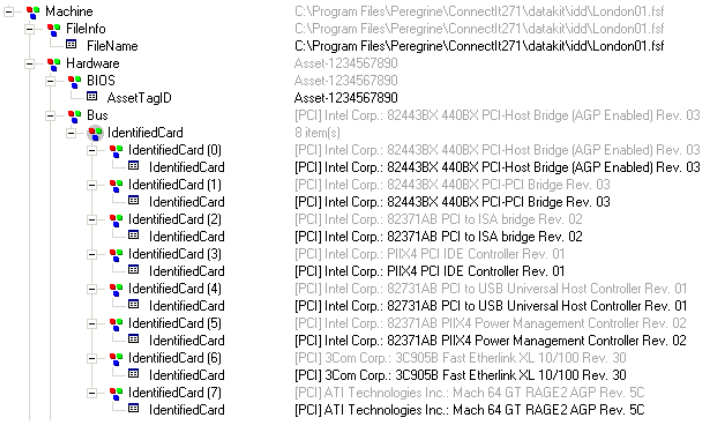
Si l'on considère le type de document suivant :



Le script :

```
Dim iCount As Integer
iCount = PifGetItemCount("Hardware.Bus.IdentifiedCard")
```

renvoie le nombre de sous-éléments pour la collection **Hardware.Bus.IdentifiedCard**. Sur le document ci-dessous, le script renvoie la valeur "8".



PifGetLongVal()

Syntaxe Basic interne

Function PifGetLongVal(strPath As String, bCkeckNodeExists As Long) As Long

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie en nombre de type entier long.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format nombre de type entier.

Exemple

```
Dim IValue As Long
IValue = PifLongVal("champ contenant un entier long")
RetVal = IValue
```

PifGetOpenSessionTime()

Syntaxe Basic interne

Function PifGetOpenSessionTime() As Date

Description

Cette fonction renvoie la date et l'heure de début de la session.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
pifLogInfoMsg("Session start time is: '" & pifGetSessionStartTime & "'")
```

PifGetParamValue()

Syntaxe Basic interne

Function PifGetParamValue(strParamName As String, strDefaultValue As String, strPath As String) As String

Description

Cette fonction est disponible pour les connecteurs AssetManagement, Base de données et ServiceCenter / Service Management, uniquement lorsque le connecteur fonctionne en mode consommation.

Cette fonction récupère la valeur contenue dans l'attribut 'Value' associé à l'attribut 'Name' dont la valeur est égale à 'strParamName' de la collection pointée par la position relative 'strPath'.

Entrée

strParamName : Valeur contenue dans l'attribut 'Name' de la collection PifParameters.

strDefaultValue : Valeur par défaut contenue dans l'attribut 'Value' de la collection PifParameters. Si aucune valeur n'est renseignée, la valeur par défaut est une chaîne vide.

strPath : Chemin relatif pour le noeud de données.

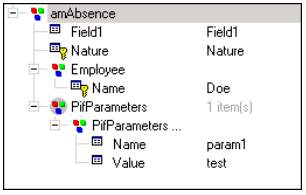
Pour naviguer dans l'arborescence du mapping, il convient d'utiliser la syntaxe ".."

Sortie

La fonction renvoie la valeur au format de type chaîne.

Exemple

La structure amAbsence contient la collection PifParameters.



Les valeurs des élements fils de la structure amAbsence sont les suivantes :

- Field1 : Field1
- Nature : Nature
- Employee.Name : Doe
- PifParameters.Name : param1
- PifParameters.Value : test

Le script de réconciliation (en mode Mise à jour) appelant cette fonction est porté par le champ Field1.

```
PifGetParamValue("param2", "notest")
```

Dans cet exemple, le chemin (valeur de 'strpath') n'est pas défini. La valeur par défaut du chemin relatif est utilisée (..PifParameters).

A la première exécution du scénario, une absence est créée pour l'employé Doe, et la valeur pour le champ 1 (Field) doit être Field 1. A la deuxième exécution, le nom de paramètre (Name) de la collection PifParameters renseigné dans la fonction n'existe pas (param2) et en conséquence la valeur par défaut spécifiée est utilisée, la valeur 'notest' renseignée dans la fonction n'est pas utilisée. Dans ce cas, la fonction renvoie la valeur par défaut 'notest'.

Remarques

Pour le bon fonctionnement de cette fonction, vous devez avoir ajouté au niveau de votre structure ou collection, la collection PifParameters.

La collection PifParameters est disponible pour chaque structure ou collection du type de document.

PifGetStringFromBlob()

Syntaxe Basic interne

Function PifGetStringFromBlob(strPath As String, bCkeckNodeExists As Long) As Long

Description

Cette fonction récupère la valeur contenue dans un blob et définie par le paramètre strPath puis la convertie en type chaîne.

Entrée

strPath : Chemin pour le noeud de données à convertir.

Pour naviguer dans l'arborescence du mapping, il convient d'utiliser la syntaxe ".."

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format de type chaîne.

Exemple

```
Dim StrValue As String
StrValue = PifGetStringFromBlob("champ contenant un blob")
RetVal = StrValue
```

PifGetStringVal()

Syntaxe Basic interne

Function PifGetStringVal(strPath As String, bCkeckNodeExists As Long) As String

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie au format chaîne.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format chaîne.

Exemple

```
Dim StrValue As String
StrValue = PifGetStringVal("amComputer.LogicalDrives(3).Name")
RetVal = StrValue
```

La syntaxe permettant de récupérer la valeur d'un champ pour un élément d'une collection est la suivante :

```
PifGetStringVal(a.b(index).c)
```

avec c comme champ, b comme collection, et index comme numéro sur lequel on veut intervenir.

Par exemple, pour le champ Name de la collection amComputer.LogicalDrives :

```
PifGetStringVal(amComputer.LogicalDrives(3).Name)
```

Cette syntaxe permet de renvoyer une valeur pour le nom (Name) du troisième disque logique (LogicalDrive(3)) de la collection.

PifGetVariantVal()

Syntaxe Basic interne

Function PifGetVariantVal(strPath As String, bCkeckNodeExists As Long) As Variant

Description

Cette fonction récupère la valeur définie par le paramètre strPath et la convertie en type variant.

Entrée

strPath : Chemin pour le noeud de données à convertir.

bCkeckNodeExists : Cette fonction permet de vérifier dans la donnée renvoyée que le noeud de données existe.

Sortie

La fonction renvoie la valeur convertie au format de type variant.

Exemple

```
strTest = RetVal = PifGetVariantVal("champ contenant un type variant")
```

PifIgnoreCollectionMapping()

Syntaxe Basic interne

Function PifIgnoreCollectionMapping(strMsg As String) As Long

Description

Cette fonction permet d'ignorer le traitement d'une collection *uniquement dans un mapping collection à collection*.

Pour mémoire, la fonction **PifIgnoreNodeMapping()** permet simplement d'ignorer l'élément courant d'une collection.

Un message d'information, contenu dans le paramètre **strMsg**, peut être envoyé dans le journal.

Entrée

- **strMsg** : Paramètre optionnel. Chaîne de caractères envoyée dans le journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

Dans l'exemple suivant, tous les éléments de la collection sont ignorés si au moins l'un des éléments possède un noeud *quantity* dont la valeur vaut '0' :

```
if [root.item(pifGetInstance).quantity] = 0 then
pifIgnoreCollectionMapping
end if
```

A titre de comparaison avec la fonction **PifIgnoreNodeMapping()**, dans l'exemple suivant, seuls les éléments de la collection possédant un noeud *quantity* dont la valeur vaut '0' sont ignorés :

```
if [root.item(pifGetInstance).quantity] = 0 then
pifIgnoreNodeMapping
end if
```

PifIgnoreDocumentMapping()

Syntaxe Basic interne

Function PifIgnoreDocumentMapping(*strMsg* As String) As Long

Description

Cette fonction permet d'ignorer le traitement d'un document.

Un message d'information, contenu dans le paramètre **strMsg**, peut être envoyé dans le journal.

Le document produit est consigné dans le journal des documents mais n'est pas transmis au prochain connecteur.

Entrée

- **strMsg** : Paramètre optionnel. Chaîne de caractères envoyée dans le journal.

Sortie

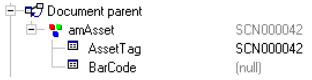
- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

Par exemple, si l'on souhaite ignorer le traitement d'un document dont le champ **BarCode** est vide, tout en consignant la valeur du champ **AssetTag** dans les lignes de suivi, on peut associer le script suivant au noeud **BarCode** :

```
If [BarCode] = "" Then
PifIgnoreDocumentMapping([AssetTag])
Else
RetVal = [BarCode]
End If
```

Si l'on utilise ce script sur le document source suivant :



le document suivant apparaît dans le journal des documents (mais n'est pas transmis au prochain connecteur) :

En outre un message d'information est consigné dans les lignes de suivi.

PifIgnoreDocumentReconc()

Syntaxe Basic interne

Function PifIgnoreDocumentReconc(strMsg As String) As Long

Description

Cette fonction permet d'ignorer un document dans une opération de réconciliation. Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Note :

Cette fonction ne doit être utilisée qu'en mode non parallélisé.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

PifIgnoreNodeMapping()

Syntaxe Basic interne

Function PifIgnoreNodeMapping(strMsg As String) As Long

Description

Cette fonction permet d'ignorer le traitement d'un noeud d'un document.

Un message d'information, contenu dans le paramètre **strMsg**, peut être envoyé dans le journal.

> Avertissement :
>
> Cette fonction ne peut pas être utilisée sur le noeud racine d'un document. Pour ignorer un document entier, utilisez la fonction **PifIgnoreDocumentMapping** .

Entrée

- **strMsg** : Paramètre optionnel. Chaîne de caractères envoyée dans le journal.

Sortie

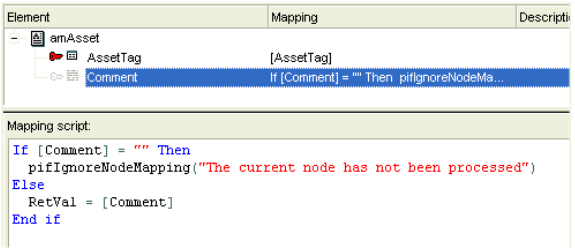
- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

Le script suivant ignore le noeud courant et enregistre un message si le champ **[Comment]** est vide.

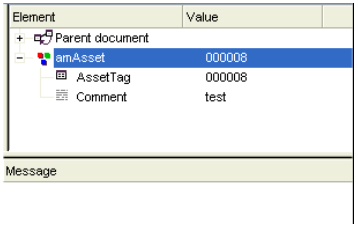
```
If [Comment] = "" Then
PifIgnoreNodeMapping("Le noeud courant n'a pas été traité")
Else
```

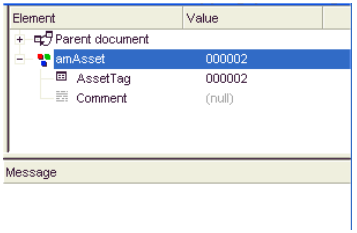
```
RetVal = [Comment]
End If
```



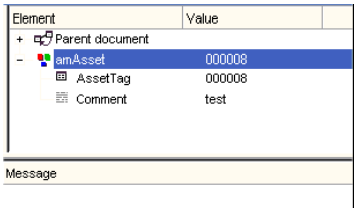
A titre d'exemple, supposons que le document source du mapping est comme suit :

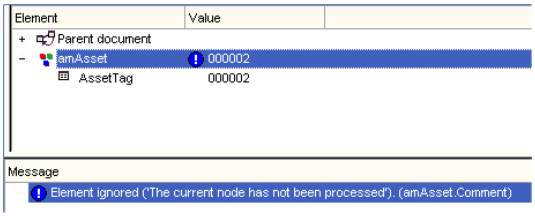
Si la fonction **PifIgnoreNodeMapping()** n'est pas utilisée, des documents comme ceux ci-dessous sont produits :





A contrario, si la fonction **PifIgnoreNodeMapping()** est utilisée, des documents comme ceux-ci sont produits :





Remarques

Note :

Le noeud n'étant pas traité, le message envoyé dans le journal est reporté sur le parent du noeud ignoré.

PifIgnoreNodeReconc()

Syntaxe Basic interne

Function PifIgnoreNodeReconc(strMsg As String) As Long

Description

Cette fonction permet d'ignorer le noeud courant (sous-document, collection, structure, ...) dans une opération de réconciliation. Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

 Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

PifIgnoreSubDocumentReconc()

Syntaxe Basic interne

Function PifIgnoreSubDocumentReconc(strMsg As String) As Long

Description

Cette fonction s'applique lorsqu'un élément de type champ est sélectionné. ELle permet, dans une opération de réconciliation, d'ignorer tout élément de même niveau que le champ sur lequel porte la fonction ainsi que l'ensemble de ses sous-éléments (liens, stuctures, collections). Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It lorsqu'un champ est sélectionné (ne s'applique par pour un élément de type lien, structure ou collection).

Pour un type de document donné de la forme :

```
Struct1
-Champ1
-Champ1bis
-Struct2
C-Champ2
```

Le champ 1bis, la structure 2 et le champ 2 sont ignorés.

PifIsInMap()

Syntaxe Basic interne

Function PifIsInMap(strKey As String, strMaptable As String, bCaseSensitive As Long) As Long

Description

Cette fonction teste si un mot-clé donné appartient à une table de correspondance. La recherche du mot-clé peut être effectuée en respectant la casse.

Entrée

- **strKey** : Nom du mot-clé sur lequel porte la recherche.
- **strMaptable** : Nom de la table de correspondance dans laquelle s'effectue la recherche.
- **0** :La recherche ne tient pas compte de la casse.

- **1** :La recherche tient compte de la casse.

bCaseSensitive : Ce paramètre précise si la recherche tient compte de la casse ou non.

Sortie

- 0 : Le mot-clé n'a pas été trouvé.
- 1 : Le mot-clé a été trouvé.

Exemple

```
If PifIsInMap("CAT_PC", "MainAsset") Then
RetVal = 1
Else
RetVal = 0
End If
```

PifLogInfoMsg()

Syntaxe Basic interne

Function PifLogInfoMsg(*strMsg* As String) As Long

Description

Cette fonction envoie un message d'information, contenu dans le paramètre **strMsg**, dans le journal.

Entrée

- **strMsg** : Chaîne de caractères contenant le message d'information à envoyer dans le journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim strBrand As String
strBrand = [DeviceBrand]
If strBrand = "" Then
PifLogInfoMsg(PifStrVal("BRAND_UNREGISTERED"))
RetVal = PifStrVal("BRAND_UNKNOWN")
Else
RetVal = strBrand
End If
```

PifLogWarningMsg()

Syntaxe Basic interne

Function PifLogWarningMsg(*strMsg* As String) As Long

Description

Cette fonction envoie un message d'avertissement, contenu dans le paramètre **strMsg**, dans le journal.

Entrée

- **strMsg** : Chaîne de caractères contenant le message d'avertissement à envoyer dans le journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
If [Brand] = "" Then
PifLogWarningMsg("Marque inconnue")
Else
RetVal = [Brand]
End if
```

PifMapValue()

Syntaxe Basic interne

Function PifMapValue(strKey As String, strMaptable As String, iPos As Long, strDefault As String, bCaseSensitive As Long, bLogErrIfOutOfRange As Long, bLogNewEntries As Long) As String

Description

Cette fonction renvoie la valeur d'un élément au sein d'une table de correspondance. L'élément est identifié de façon unique au moyen des paramètres **strKey**, **strMapTable**, **iPos**.

La recherche de l'élément peut tenir compte ou non de la casse.

Entrée

- **strKey** : Mot-clé permettant d'identifier la ligne de la table de correspondance qui contient l'élément.
- **strMaptable** : Nom de la table de correspondance dans laquelle s'effectue la recherche.
- **iPos** : Numéro de la colonne dans laquelle se trouve l'élément dont on veut récupérer la valeur.

  Note :

  Ce paramètre peut prendre toute valeur supérieure ou égale à 0; 0 correspondant à la première colonne.

- **strDefault** : Valeur par défaut renvoyée si l'élément n'est pas trouvé.
- **bCaseSensitive** : Ce paramètre précise si la recherche tient compte de la casse ou non.
  - **0** : La recherche ne tient pas compte de la casse.
  - **1** (valeur par défaut) : La recherche tient compte de la casse.
- **bLogErrIfOutOfRange** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsque le numéro de colonne (paramètre **iPos**) est inférieur à 0 ou supérieur au nombre de colonnes de la table de correspondance.

> **Note :**
>
> Par défaut, ce paramètre prend la valeur TRUE.

- **bLogNewEntries** : Ce paramètre permet de créer dans un nouveau fichier attenant au scénario la table de correspondance pour les clés trouvées et non déclarées au préalable dans une table de correspondance. Ce fichier porte le nom du scénario et est préfixé avec _NewMapKeys.mpt

> **Note :**
>
> Les tables de correspondance relatives au scénario ne sont pas automatiquement mises à jour. Pour effectuer une mise à jour des tables de correspondance, éditez le fichier nouvellement créé, et copiez-collez les nouvelles informations dans le fichier des tables de correspondance.

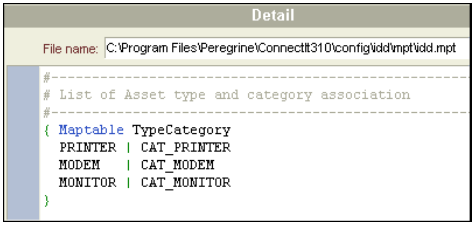
  - **0** (valeur par défaut) : La création automatique du fichier de tables de correspondance n'est pas activée.
  - **1** : La création automatique du fichier de tables de correspondance est activée.

Sortie

La fonction renvoie la chaîne trouvée ou la chaîne par défaut (celle contenue dans le paramètre **strDefault**) si l'élément n'est pas trouvé.

Exemple

En prenant pour exemple la table de correspondance suivante :



Le code suivant :

```
pifMapValue("PRINTER", "TypeCategory", 1, "")
```

retourne la valeur **CAT_PRINTER**.

Le code suivant :

```
pifMapValue("Printer", "TypeCategory", 1, "")
```

retourne la valeur par défaut. En effet, la recherche tenant compte de la casse, l'élément recherché n'est pas trouvé.

Le code suivant :

```
pifMapValueContaining("Print", "TypeCategory", 1, 1, 1 "")
```

retourne la valeur par défaut, enregistre une erreur au journal des événements et créé un fichier de tables de correspondance.

Remarques

Note :

Cette fonction tient compte des caractères joker suivants :

- ? : correspond à n'importe quel caractère.
- * : correspond à un nombre indéfini de caractères quelconques.

PifMapValueContaining()

Syntaxe Basic interne

Function PifMapValueContaining(strKey As String, strMaptable As String, iPos As Long, strDefault As String, bCaseSensitive As Long, bLogErrIfOutOfRange As Long, bLogNewEntries As Long) As String

Description

Cette fonction renvoie la valeur d'un élément au sein d'une table de correspondance. L'élément est identifié de façon unique au moyen des paramètres **strKey**, **strMapTable**, **iPos**.

La recherche de l'élément peut tenir compte ou non de la casse.

A la difference de la fonction **PifMapValue**, le paramètre **strKey** peut être un sous-ensemble du mot-clé recherché.

Par exemple si **strKey** contient "Moniteur", l'élément de mot-clé "Cat_Moniteur" sera trouvé.

Entrée

- **strKey** : Expression contenant une des clés définie dans la table de correspondance.
- **strMaptable** : Nom de la table de correspondance dans laquelle s'effectue la recherche.
- **iPos** : Numéro de la colonne dans laquelle se trouve l'élément dont on veut récupérer la valeur.

> Note :
>
> Ce paramètre peut prendre toute valeur supérieure ou égale à 0; 0 correspondant à la première colonne.

- **strDefault** : Valeur par défaut renvoyée si l'élément n'est pas trouvé.
- - **0** : La recherche ne tient pas compte de la casse.
  - **1** (valeur par défaut): La recherche tient compte de la casse.

  bCaseSensitive : Ce paramètre précise si la recherche tient compte de la casse ou non.
- **bLogErrIfOutOfRange** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsque le numéro de colonne (paramètre **iPos**) est inférieur à 0 ou supérieur au nombre de colonnes de la table de correspondance.

> Note :
>
> Par défaut, ce paramètre prend la valeur TRUE.

- **bLogNewEntries** : Ce paramètre permet de créer dans un nouveau fichier attenant au scénario la table de correspondance pour les clés trouvées et non déclarées au préalable dans une table de correspondance. Ce fichier porte le nom du scénario et est préfixé avec _NewMapKeys.mpt

> Note :
>
> Les tables de correspondance relatives au scénario ne sont pas automatiquement mises à jour. Pour effectuer une mise à jour des tables de correspondance, éditez le fichier nouvellement créé, et copiez-collez les nouvelles informations dans le fichier des tables de correspondance.

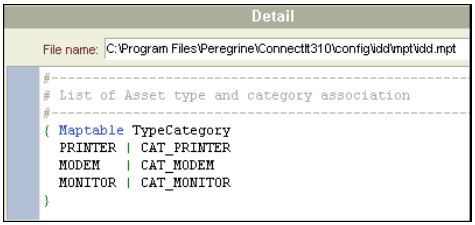
  - **0** (valeur par défaut) : La création automatique du fichier de tables de correspondance n'est pas activée.
  - **1** : La création automatique du fichier de tables de correspondance est activéé.

Sortie

La fonction renvoie la chaîne trouvée ou la chaîne par défaut (celle contenue dans le paramètre **strDefault**) si l'élément n'est pas trouvé.

Exemple

En prenant pour exemple la table de correspondance suivante :



Le code suivant :

```
pifMapValueContaining("PRINT", "TypeCategory", 1, "")
```

retourne la valeur **CAT_PRINTER**.

Note :

Si, dans la table de correspondance, plusieurs clés correspondent au paramètre **StrKey**, alors la première clé rencontrée sera renvoyée par la fonction.

Le code suivant :

```
pifMapValueContaining("Print", "TypeCategory", 1, "")
```

retourne la valeur par défaut. En effet, la recherche tenant compte de la casse, l'élément recherché n'est pas trouvé.

Le code suivant :

```
pifMapValueContaining("Print", "TypeCategory", 1, 1, 1 "")
```

retourne la valeur par défaut, enregistre une erreur au journal des événements et créé un fichier de tables de correspondance.

Remarques

Note :

Cette fonction tient compte des caractères joker suivants :

- ? : correspond à n'importe quel caractère.
- * : correspond à un nombre indéfini de caractères quelconques.

PifMapValueContainingEx()

Syntaxe Basic interne

Function PifMapValueContainingEx(strKey As String, strMaptable As String, strColName As String, strDefault As String, bCaseSensitive As Long, bLogErrIfOutOfRange As Long, bLogNewEntries As Long) As String

Description

Cette fonction renvoie la valeur d'un élément au sein d'une table de correspondance. L'élément est identifié de façon unique au moyen des paramètres **strKey**, **strMapTable**, **strColName**.

La recherche de l'élément peut tenir compte ou non de la casse.

A la difference de la fonction **PifMapContaining**, le paramètre **strColName** définit le nom de la colonne au lieu de sa position.

Cette fonction est utile lorsque :

- la table de correspondance a été créée à partir d'un format (les noms de colonne correspondent aux noms des éléments du format)
- les noms des colonnes a été défini manuellement dans les tables de correspondance à l'aide de l'éditeur de tables de correspondance.

Entrée

- **strKey** : Expression contenant une des clés définie dans la table de correspondance.
- **strMaptable** : Nom de la table de correspondance dans laquelle s'effectue la recherche.

- **strColName** : Nom de la colonne dans laquelle se trouve l'élément dont on veut récupérer la valeur
- **strDefault** : Valeur par défaut renvoyée si l'élément n'est pas trouvé.
- - **0** : La recherche ne tient pas compte de la casse.
  - **1** (valeur par défaut): La recherche tient compte de la casse.

  bCaseSensitive : Ce paramètre précise si la recherche tient compte de la casse ou non.
- **bLogErrIfOutOfRange** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsque le numéro de colonne (paramètre **iPos**) est inférieur à 0 ou supérieur au nombre de colonnes de la table de correspondance.

   Note :

  Par défaut, ce paramètre prend la valeur TRUE.
- **bLogNewEntries** : Ce paramètre permet de créer dans un nouveau fichier attenant au scénario la table de correspondance pour les clés trouvées et non déclarées au préalable dans une table de correspondance. Ce fichier porte le nom du scénario et est préfixé avec _NewMapKeys.mpt

   Note :

  Les tables de correspondance relatives au scénario ne sont pas automatiquement mises à jour. Pour effectuer une mise à jour des tables de correspondance, éditez le fichier nouvellement créé, et copiez-collez les nouvelles informations dans le fichier des tables de correspondance.

  - **0** (valeur par défaut) : La création automatique du fichier de tables de correspondance n'est pas activée.
  - **1** : La création automatique du fichier de tables de correspondance est activéé.

Sortie

La fonction renvoie la chaîne trouvée ou la chaîne par défaut (celle contenue dans le paramètre **strDefault**) si l'élément n'est pas trouvé.

Remarques

Note :

Cette fonction tient compte des caractères joker suivants :

- ? : correspond à n'importe quel caractère.
- * : correspond à un nombre indéfini de caractères quelconques.

Si aucune table de correspondance n'a été créée à partir d'un format ou manuellement à l'aide de l'éditeur de table de correspondances, il convient d'utiliser les fonctions **PifMapValueContaining** ou **PifMapValue**.

PifMapValueEx()

Syntaxe Basic interne

Function PifMapValueEx(strKey As String, strMaptable As String, strColName As String, strDefault As String, bCaseSensitive As Long, bLogErrIfOutOfRange As Long, bLogNewEntries As Long) As String

Description

Cette fonction renvoie la valeur d'un élément au sein d'une table de correspondance. L'élément est identifié de façon unique au moyen des paramètres **strKey**, **strMapTable**, **srtColName**. Cette fonction se distingue de la fonction **PIFMAPVALUE** par le paramètre **strColName** qui spécifie le nom de la colonne au lieu de son numéro.

La recherche de l'élément peut tenir compte ou non de la casse.

Cette fonction est utile lorsque :

- la table de correspondance a été créée à partir d'un format (les noms de colonne correspondent aux noms des éléments du format)
- les noms des colonnes a été défini manuellement dans les tables de correspondance à l'aide de l'éditeur de tables de correspondance.

Entrée

- **strKey** : Mot-clé permettant d'identifier la ligne de la table de correspondance qui contient l'élément.

- **strMaptable** : Nom de la table de correspondance dans laquelle s'effectue la recherche.
- **strColName** : Nom de la colonne dans laquelle se trouve l'élément dont on veut récupérer la valeur
- **strDefault** : Valeur par défaut renvoyée si l'élément n'est pas trouvé.
- **bCaseSensitive** : Ce paramètre précise si la recherche tient compte de la casse ou non.
  - **0** : La recherche ne tient pas compte de la casse.
  - **1** (valeur par défaut) : La recherche tient compte de la casse.
- **bLogErrIfOutOfRange** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsque le numéro de colonne (paramètre **iPos**) est inférieur à 0 ou supérieur au nombre de colonnes de la table de correspondance.

  Note :

  Par défaut, ce paramètre prend la valeur TRUE.

- **bLogNewEntries** : Ce paramètre permet de créer dans un nouveau fichier attenant au scénario la table de correspondance pour les clés trouvées et non déclarées au préalable dans une table de correspondance. Ce fichier porte le nom du scénario et est préfixé avec _NewMapKeys.mpt

  Note :

  Les tables de correspondance relatives au scénario ne sont pas automatiquement mises à jour. Pour effectuer une mise à jour des tables de correspondance, éditez le fichier nouvellement créé, et copiez-collez les nouvelles informations dans le fichier des tables de correspondance.

  - **0** (valeur par défaut) : La création automatique du fichier de tables de correspondance n'est pas activée.
  - **1** : La création automatique du fichier de tables de correspondance est activéé.

Sortie

La fonction renvoie la chaîne trouvée ou la chaîne par défaut (celle contenue dans le paramètre **strDefault**) si l'élément n'est pas trouvé.

Remarques

Note :

Cette fonction tient compte des caractères joker suivants :

- ? : correspond à n'importe quel caractère.
- * : correspond à un nombre indéfini de caractères quelconques.

Si aucune table de correspondance n'a été créée à partir d'un format ou manuellement à l'aide de l'éditeur de table de correspondances, il convient d'utiliser les fonctions **PifMapValueContaining** ou **PifMapValue**.

PifNewQueryFromFmtName()

Syntaxe Basic interne

Function PifNewQueryFromFmtName(strCntrName As String, strFmtName As String, strLayer As String) As Long

Description

Cette fonction crée une requête sur un type de document préalablement défini dans la liste des documents produits par une ressource.

Entrée

- **strCntrName** : Ce paramètre contient le nom de la ressource (celle sur laquelle la requête est effectuée).
- **strFmtName** : Ce paramètre contient l'identifiant du type de document (préalablement défini comme type de document produit).
- **strLayer**Ce paramètre permet de définir une directive de production pour un type de document produit.

  La directive de production peut être :
  - une clause WHERE respectant la syntaxe propre au connecteur
  - une description XML définissant les directives de production applicables pour le connecteur, comme par exemple une clause Order By. La syntaxe XML des directives de production est décrite dans la documentation de **PifNewQueryFromXML**.

> **Note :**
>
> Si le type de document **strFmtName** contient déjà des directives de production, elles seront fusionnées avec celles définies dans le paramètre **strLayer**. Cependant, si les deux définissent une valeur différente pour la même directive, la valeur définie dans **strLayer** aura la précédence.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple ci-dessous spécifie une simple clause Where pour le paramètre **strLayer** :

```
dim hQuery as long
dim iRc as long

hQuery = pifNewQueryFromFmtName("Asset Management", "amEmplDept", "Name li
ke 'A%'")

Dim strValue as string

while (iRc = 0)
iRc = pifQueryNext(hQuery)
if iRc = 0 then
strValue = pifQueryGetStringVal(hQuery, "Name")
piflogInfoMsg strValue
end if
wend

iRc = pifQueryClose(hQuery)
```

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifQueryClose() [page 159]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]

- PifQueryGetStringVal() [page 165]

PifNewQueryFromXml()

Syntaxe Basic interne

Function PifNewQueryFromXml(strCntrName As String, strQuery As String, strLayer As String) As Long

Description

Cette fonction crée une requête pour une ressource. Le type de document doit être intégralement défini sous forme XML par le paramètre **strQuery**. Le traitement (clause d'une requête AQL) est défini sous forme XML par le paramètre **strLayer**.

Entrée

- **strCntrName** : Ce paramètre contient le nom de la ressource (celle sur laquelle la requête est effectuée).
- **strQuery** : Ce paramètre contient un document XML qui définit le type de document (attributs, structures, collections) sur lequel s'effectue la requête.
- **strLayer** : Ce paramètre contient la clause Where d'une requête AQL ou un document XML qui définit les directives de production (clause Where, Order By, ...) pour la requête.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
strLayer =             "<Directives>"
strLayer = strLayer +   "<Where>Name = '" + GetXmlElementValue([Name]) + "
'</Where>"
strLayer = strLayer +   "<OrderBy>BarCode</OrderBy>"
strLayer = strLayer +      "<Where Path='ItemsUsed'>AssetTag like 'A%'</Wh
ere>"
strLayer = strLayer + "</Directives>"
```

Remarques

Dans cet exemple, on construit un document XML qui spécifie à la fois le document à produire et les clauses de la requête exécutée. Comme pour tout document XML, il doit être valide. Les caractères interdits (par exemple <, &...) doivent donc être échappés à l'aide des fonctions GetXmlElementValue() [page 87] et GetXmlAttributeValue() [page 86].

Note :

Lorsque **strLayer** contient une simple clause Where AQL, l'utilisation de GetXmlElementValue() [page 87] est inutile.

La description précédente suppose que vous connaissez le type de document à produire et que la requête contient une simple clause WHERE. L'exemple ci-dessous illustre l'utilisation de la fonction sans connaître le type de document à produire. En outre, il explicite l'utilisation d'autre clauses AQL.

```
dim hQuery as long
dim strQuery as string
dim strLayer as string
dim iRc as long

strQuery =             "<STRUCTURE Name='amEmplDept'>"
strQuery = strQuery +   "<ATTRIBUTE Name='Name'/>"
strQuery = strQuery +   "<ATTRIBUTE Name='BarCode'/>"
strQuery = strQuery +   "<COLLECTION Name='ItemsUsed'>"
strQuery = strQuery +     "<ATTRIBUTE Name='AssetTag'/>"
strQuery = strQuery +   "</COLLECTION>"
strQuery = strQuery + "</STRUCTURE>"

strLayer =             "<Directives>"
strLayer = strLayer +   "<Where>Name = 'Taltek'</Where>"
strLayer = strLayer +   "<OrderBy>BarCode</OrderBy>"
strLayer = strLayer +      "<Where Path='ItemsUsed'>AssetTag like 'A%'</Wh
ere>"
strLayer = strLayer + "</Directives>"

hQuery = pifNewQueryFromXml("Asset Management", strQuery, strLayer)

Dim strValue as string

while (iRc = 0)
iRc = pifQueryNext(hQuery)
if iRc = 0 then
strValue = pifQueryGetStringVal(hQuery, "Name")
piflogInfoMsg strValue
end if
wend

iRc = pifQueryClose(hQuery)
```

Note :

Dans cet exemple, on construit un document XML qui spécifie à la fois le document à produire et les clauses de la requête exécutée. Comme pour tout document XML, il doit être valide. Les caractères interdits (par exemple <,&...) doivent donc être remplacés par les entités correspondantes (<,&...).

Voir aussi :

- PifQueryClose() [page 159]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]
- PifQueryGetStringVal() [page 165]

PifNodeExists()

Syntaxe Basic interne

Function PifNodeExists(strPath As String) As Long

Description

Cette fonction permet de tester si un noeud, identifié par son chemin d'accès complet, existe au sein d'un document produit.

Entrée

- **strPath** : Chemin d'accès complet du noeud concerné par l'opération.

Sortie

La fonction renvoie l'une des valeurs suivantes :

- **0** :si le noeud n'existe pas.
- **1** :si le noeud existe

Exemple

```
If PifNodeExists("Hardware.Peripherials.Printer") = 0 Then
PifIgnoreNodeMapping
End If
```

PifOpenODBCDatabase()

Syntaxe Basic interne

Function PifOpenODBCDatabase(strDSN As String, strLogin As String, strPwd As String) As Long

Description

Cette fonction permet d'ouvrir une connexion à une base de données ODBC.

Entrée

- **strDSN** : Nom de la source de données de la connexion ODBC.
- **strLogin** : Login pour la connexion à la base de données ODBC.
- **strPwd** : Mot de passe associé au login (**strLogin**) pour la connexion à la base de données ODBC.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim iRet As Integer
iRet = PifOpenOdbcDatabase(PifStrVal("DSN"), PifStrVal("LOGIN"), PifStrVal
("PWD"))

if  iRet = 0 Then
Dim strQuery As String
strQuery = "SELECT a.AssetName, a.FullName FROM Asset a"
iRet = PifCreateDynaMapTable(PifStrVal("DSN"), "Assets", strQuery, FALSE)
End If
```

```
Dim strRes as String
if  iRet = 0 Then
strRes = PifMapValue([FullName], "Assets", 1, "", 1)
End If

If iRet = 0 and strRes = "" Then
iRet = PifExecODBCSql(PifStrVal("DSN"), "INSERT INTO Asset(AssetNAme, Asse
tFullName)
```

Remarques

Note :

Si une connexion existe déjà sur la source de données, la fonction n'en ouvre pas une seconde.

PifQueryClose()

Syntaxe Basic interne

Function PifQueryClose(lQueryHandle As Long) As Long

Description

Cette fonction ferme la requête et libère toutes les ressources internes utilisées par la requête.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
dim hQuery as long
dim iRc as long

hQuery = pifNewQueryFromFmtName("Asset Management", "amEmplDept", "Name li
ke 'A%'")

Dim strValue as string

while (iRc = 0)
iRc = pifQueryNext(hQuery)
if iRc = 0 then
strValue = pifQueryGetStringVal(hQuery, "Name")
piflogInfoMsg strValue
end if
wend

iRc = pifQueryClose(hQuery)
```

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]
- PifQueryGetStringVal() [page 165]

PifQueryGetDateVal()

Syntaxe Basic interne

Function PifQueryGetDateVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Date

Description

Cette fonction renvoie la valeur (sous la forme d'une Date) d'un noeud du document courant. Le document courant est celui sur lequel le curseur de la requête a été fixé au moyen de la fonction **PifQueryNext()**.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.
- **strPath** : Ce paramètre contient le chemin du noeud du document courant dont vous souhaitez récupérer la valeur.
- **bPathMustExist** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsqu'aucune valeur n'est récupérée pour le chemin spécifié dans le paramètre **strPath**.

Sortie

Valeur du noeud identifié

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryClose() [page 159]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]
- PifQueryGetStringVal() [page 165]

PifQueryGetDoubleVal()

Syntaxe Basic interne

Function PifQueryGetDoubleVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Double

Description

Cette fonction renvoie la valeur (sous la forme d'un Double) d'un noeud du document courant. Le document courant est celui sur lequel le curseur de la requête a été fixé au moyen de la fonction **PifQueryNext()**.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.
- **strPath** : Ce paramètre contient le chemin du noeud du document courant dont vous souhaitez récupérer la valeur.
- **bPathMustExist** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsqu'aucune valeur n'est récupérée pour le chemin spécifié dans le paramètre **strPath**.

Sortie

Valeur du noeud identifié

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryClose() [page 159]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]

- PifQueryGetStringVal() [page 165]

PifQueryGetIntVal()

Syntaxe Basic interne

Function PifQueryGetIntVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Long

Description

Cette fonction renvoie la valeur (sous la forme d'un Entier) d'un noeud du document courant. Le document courant est celui sur lequel le curseur de la requête a été fixé au moyen de la fonction **PifQueryNext()**.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.
- **strPath** : Ce paramètre contient le chemin du noeud du document courant dont vous souhaitez récupérer la valeur.
- **bPathMustExist** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsqu'aucune valeur n'est récupérée pour le chemin spécifié dans le paramètre **strPath**.

Sortie

Valeur du noeud identifié

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]

- PifQueryGetDoubleVal() [page 162]
- PifQueryClose() [page 159]
- PifQueryGetLongVal() [page 164]
- PifQueryGetStringVal() [page 165]

PifQueryGetLongVal()

Syntaxe Basic interne

Function PifQueryGetLongVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Long

Description

Cette fonction renvoie la valeur (sous la forme d'un Long) d'un noeud du document courant. Le document courant est celui sur lequel le curseur de la requête a été fixé au moyen de la fonction **PifQueryNext()**.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.
- **strPath** : Ce paramètre contient le chemin du noeud du document courant dont vous souhaitez récupérer la valeur.
- **bPathMustExist** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsqu'aucune valeur n'est récupérée pour le chemin spécifié dans le paramètre **strPath**.

Sortie

Valeur du noeud identifié

Exemple

```
strValue = PifQueryGetLongVal(hQuery, "Name")
```

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryClose() [page 159]
- PifQueryGetStringVal() [page 165]

PifQueryGetStringVal()

Syntaxe Basic interne

Function PifQueryGetStringVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As String

Description

Cette fonction renvoie la valeur (sous la forme d'une chaîne) d'un noeud du document courant. Le document courant est celui sur lequel le curseur de la requête a été fixé au moyen de la fonction **PifQueryNext()**.

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.
- **strPath** : Ce paramètre contient le chemin du noeud du document courant dont vous souhaitez récupérer la valeur.
- **bPathMustExist** : En fonction de la valeur de ce paramètre, une erreur sera générée (=TRUE) ou non (=FALSE) dans le journal des événements lorsqu'aucune valeur n'est récupérée pour le chemin spécifié dans le paramètre **strPath**.

Sortie

Valeur du noeud identifié

Remarques

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryNext() [page 166]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]
- PifQueryClose() [page 159]

PifQueryNext()

Syntaxe Basic interne

Function PifQueryNext(lQueryHandle As Long) As Long

Description

Cette fonction fixe le curseur de la requête sur le prochain résultat. Le curseur n'est pas fixé sur le premier document après un appel aux fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**. L'utilisateur doit faire appel à la fonction **PifQueryNext()** pour avoir accès aux valeurs du document courant avec une des fonctions suivantes :

- **PifQueryGetStringVal()**
- **PifQueryGetDateVal()**
- **PifQueryGetDoubleVal()**
- **PifQueryGetLongVal()**
- **PifQueryGetIntVal()**

Entrée

- **lQueryHandle** : Ce paramètre contient un descripteur sur la requête créée au moyen des fonctions **PifNewQueryFromFmtName()** ou **PifNewQueryFromXml()**.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
dim hQuery as long
dim iRc as long

hQuery = pifNewQueryFromFmtName("Asset Management", "amEmplDept", "Name li
ke 'A%'")

Dim strValue as string

while (iRc = 0)
iRc = pifQueryNext(hQuery)
if iRc = 0 then
strValue = pifQueryGetStringVal(hQuery, "Name")
piflogInfoMsg strValue
end if
wend

iRc = pifQueryClose(hQuery)
```

Remarques

La fonction sort en erreur lorsqu'il n'y a plus de données à parcourir (code d'erreur -2003).

Voir aussi :

- PifNewQueryFromXml() [page 155]
- PifNewQueryFromFmtName() [page 153]
- PifQueryClose() [page 159]
- PifQueryGetDateVal() [page 160]
- PifQueryGetDoubleVal() [page 162]
- PifQueryGetIntVal() [page 163]
- PifQueryGetLongVal() [page 164]
- PifQueryGetStringVal() [page 165]

PifRejectCollectionMapping()

Syntaxe Basic interne

Function PifRejectCollectionMapping(*strMsg* As String) As Long

Description

Cette fonction permet de rejeter tous les éléments d'une collection, *uniquement dans un mapping collection à collection*.

Pour mémoire, la fonction **PifRejectNodeMapping()** permet simplement de rejeter le noeud courant d'un document.

Un message, contenu dans le paramètre **strMsg** est envoyé dans le fichier journal.

Note :

L'utilisation de cette fonction équivaut à un rejet partiel du document. Les rejets partiels peuvent être récupérés dans les bilans de traitement de la boîte de mapping.

Entrée

- **strMsg** : Ce paramètre optionnel contient le message à inscrire dans le fichier journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

Dans l'exemple suivant, tous les éléments de la collection sont rejetés si au moins l'un des éléments possède un noeud *quantity* dont la valeur vaut '-1' :

```
if [root.item(pifGetInstance).quantity] = -1 then
pifRejectCollectionMapping("invalid quantity")
end if
```

A titre de comparaison avec la fonction **PifRejectNodeMapping()**, dans l'exemple suivant, seuls les éléments de la collection possédant un noeud *quantity* dont la valeur vaut '-1' sont rejetés :

```
if [root.item(pifGetInstance).quantity] = -1 then
pifRejectNodeMapping
end if
```

Remarques

Voir aussi :

- PifRejectNodeMapping() [page 171]
- PifRejectDocumentMapping() [page 169]

PifRejectDocumentMapping()

Syntaxe Basic interne

Function PifRejectDocumentMapping(strMsg As String) As Long

Description

Cette fonction permet de rejeter un document.

Un message d'information, contenu dans le paramètre **strMsg**, peut être envoyé dans le journal.

Le document n'est pas transmis au prochain connecteur.

Entrée

- **strMsg** : Paramètre optionnel. Chaîne de caractères envoyée dans le journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim strNetAddress As String
strNetAddress = [Hardware.TCPIP.PhysicalAddress]

If strNetAddress = "" Then
PifRejectDocumentMapping("Document rejected: missing MAC address")
Else
RetVal = strNetAddress
End If
```

PifRejectDocumentReconc()

Syntaxe Basic interne

Function PifRejectDocumentReconc(strMsg As String) As Long

Description

Cette fonction permet de rejeter un document dans une opération de réconciliation. Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Note :

Cette fonction ne doit être utilisée qu'en mode non parallélisé.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

Voir aussi :

- PifRejectNodeReconc() [page 172]
- PifRejectSubDocumentReconc() [page 173]

PifRejectNodeMapping()

Syntaxe Basic interne

Function PifRejectNodeMapping(*strMsg* As String) As Long

Description

Cette fonction permet de rejeter le noeud courant d'un document.

Un message, contenu dans le paramètre **strMsg** est envoyé dans le fichier journal.

Note :

L'utilisation de cette fonction équivaut à un rejet partiel du document. Les rejets partiels peuvent être récupérés dans les bilans de traitement de la boîte de mapping.

Entrée

- **strMsg** : Paramètre optionnel. Chaîne de caractères envoyée dans le journal.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
If [Location.Name] = "" Then
PifRejectNodeMapping(PifStrVal("UNKNOWN_LOCATION"))
End If
```

Remarques

Avertissement :

Cette fonction ne peut pas être utilisée sur le noeud racine d'un document. Pour rejeter un document entier, utilisez la fonction **PifRejectDocumentMapping** .

PifRejectNodeReconc()

Syntaxe Basic interne

Function PifRejectNodeReconc(strMsg As String) As Long

Description

Cette fonction permet de rejeter le noeud courant (sous-document, collection, structure, ...) dans une opération de réconciliation. Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

PifRejectSubDocumentReconc()

Syntaxe Basic interne

Function PifRejectSubDocumentReconc(strMsg As String) As Long

Description

Cette fonction permet de rejeter un sous-document dans une opération de réconciliation. Aucune insertion ni mise à jour n'est effectuée. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

PifScenarioPath()

Syntaxe Basic interne

Function PifScenarioPath() As String

Description

Cette fonction renvoie le chemin complet d'un fichier scénario. Dans le cas où le scénario n'a pas encore été sauvegardé, la fonction renvoie une valeur vide.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strFile
strFile = pifScenarioPath & "Format.xml"

Open strFile For Output As #1
Print #1, pifXmlDump
Close #1
```

PifSetDateVal()

Syntaxe Basic interne

Function PifSetDateVal(strPath As String, dttVal As Date) As Long

Description

Cette fonction permet de fixer la valeur d'un noeud dans le document cible. Si le noeud n'existe pas, la fonction le crée.

Entrée

- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.
- **dttVal** : Ce paramètre contient la valeur (date) que vous souhaitez affecter au noeud.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim dtCurrent as Date
Dim lRet as Long
dtCurrent = Date()
lRet = PifSetDateVal("ValueDate", dtCurrent)
```

Remarques

Voir aussi :

- PifSetDoubleVal() [page 175]
- PifSetLongVal() [page 176]
- PifSetNullVal() [page 177]
- PifSetStringVal() [page 181]

PifSetDoubleVal()

Syntaxe Basic interne

Function PifSetDoubleVal(strPath As String, dVal As Double) As Long

Description

Cette fonction permet de fixer la valeur d'un noeud dans le document cible. Si le noeud n'existe pas, la fonction le crée.

Entrée

- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.
- **dVal** : Ce paramètre contient la valeur (double) que vous souhaitez affecter au noeud.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim d as Double
Dim lRet as Long
d = 2.5
lRet = PifSetDoubleVal("ValueDouble", d)
```

Remarques

Voir aussi :

- PifSetDateVal() [page 174]
- PifSetLongVal() [page 176]
- PifSetNullVal() [page 177]
- PifSetStringVal() [page 181]

PifSetLongVal()

Syntaxe Basic interne

Function PifSetLongVal(strPath As String, lVal As Long) As Long

Description

Cette fonction permet de fixer la valeur d'un noeud dans le document cible. Si le noeud n'existe pas, la fonction le crée.

Entrée

- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.
- **lVal** : Ce paramètre contient la valeur (entier long) que vous souhaitez affecter au noeud.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim l as Long
Dim lRet as Long
l = 2
lRet = PifSetLongVal("ValueLong", l)
```

Remarques

Voir aussi :

- PifSetDoubleVal() [page 175]
- PifSetDateVal() [page 174]
- PifSetNullVal() [page 177]
- PifSetStringVal() [page 181]

PifSetNullVal()

Syntaxe Basic interne

Function PifSetNullVal(strPath As String) As Long

Description

Cette fonction permet de renseigner un noeud du document cible avec la valeur 'NULL'. Si le noeud n'existe pas, la fonction procède à sa création.

Entrée

- **strPath** : Ce paramètre contient le nom complet du noeud concerné par l'opération. Si ce paramètre est omis ou vide, le noeud courant est sélectionné.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
If [Name] = "" Then
PifSetNullVal("FirstName")
End if
```

Remarques

Voir aussi :

- PifSetDoubleVal() [page 175]
- PifSetLongVal() [page 176]
- PifSetDateVal() [page 174]
- PifSetStringVal() [page 181]

PifSetParamValue()

Syntaxe Basic interne

Function PifSetParamValue(strParamName As String, strValue As String, strPath As String)

Description

Cette fonction est disponible pour les connecteurs AssetManagement, Base de données et ServiceCenter / Service Management, uniquement lorsque le connecteur fonctionne en mode consommation.

Cette fonction permet de définir une valeur de type texte accessible dans un script de mapping ou un script de réconciliation.

Entrée

strParamName : Valeur contenue dans l'attribut 'Name' de la collection PifParameters.

strValue : Valeur contenue dans l'attribut 'Value' de la collection PifParameters.

strPath : Chemin relatif pour le noeud de données.

Les noeuds de la collection doivent avoir été créés et une valeur renseignée pour l'attribut 'value'.

Pour naviguer dans l'arborescence du mapping, il convient d'utiliser la syntaxe ".."

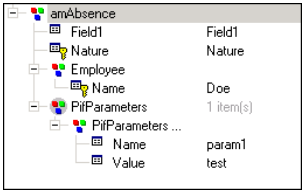
Sortie

La fonction renvoie la valeur au format de type chaîne.

A la deuxième exécution du scénario, le champ 1 garde sa valeur (car vOldVal=vNewVal='Field1'). Le script de réconciliation change le paramètre 'param1' en lui attribuant la valeur 'newvalue' (alors que la valeur initiale était 'test').

Exemple

La structure amAbsence contient la collection PifParameters.



Les valeurs des élements fils de la structure amAbsence sont les suivantes :

- Field1 : Field1
- Nature : Nature
- Employee.Name : Doe
- PifParameters.Name : param1
- PifParameters.Value : test

Le script de réconciliation (en mode Mise à jour) appelant cette fonction est porté par le champ Nature.

```
call PifSetParamValue("param1", "newValue")
RetVal = vNewVal
```

A la première exécution du scénario, une absence est créée pour l'employé Doe, la valeur pour le champ 1 est 'Field 1' et la valeur pour le champ Nature est 'nature'.

Remarques

Pour le bon fonctionnement de cette fonction, vous devez avoir ajouté au niveau de votre structure ou collection, la collection PifParameters.

La collection PifParameters est disponible pour chaque structure ou collection du type de document, en mode consommation.

PifSetPendingDocument()

Syntaxe Basic interne

Function PifSetPendingDocument(*strMsg* As String) As Long

Description

Cette fonction permet de mettre un document en attente lors d'une opération de réconciliation. Le document ne sera pas inséré ou mis à jour. Un message d'information, contenu dans le paramètre **strMsg** peut être affiché dans la trace.


Note :

Cette fonction ne doit être utilisée qu'en mode non parallélisé.

Entrée

- **strMsg** : Ce paramètre contient le message d'information affiché dans la trace.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
If vNewVal >= vOldVal Then
RetVal = vNewVal
Else
RetVal = vOld
PifSetPendingDocument(PifStrVal("RECONC_PROPOSAL_NOT_VALIDATED"))
End If
```

Remarques

Note :

Cette fonction est disponible uniquement dans les scripts de réconciliation de Connect-It.

PifSetStringVal()

Syntaxe Basic interne

Function PifSetStringVal(strPath As String, strVal As String) As Long

Description

Cette fonction permet de fixer la valeur d'un noeud dans le document cible. Si le noeud n'existe pas, la fonction le crée.

Entrée

- **strPath** : Ce paramètre contient le chemin complet du noeud concerné par l'opération.
- **strVal** : Ce paramètre contient la valeur (chaîne de caractères) que vous souhaitez affecter au noeud.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
Dim str as String
Dim lRet as Long
str = "100.10.1.1"
lRet = PifSetStringVal("ipaddress", str)
```

Note :

Il est impératif de récupérer le code de retour pour la fonction. Dans le cas contraire, une erreur de compilation est renvoyée.

La syntaxe permettant de fixer la valeur d'un champ pour un élément d'une collection est la suivante :

```
PifSetStringVal(a.b(index).c)
```

avec c comme champ, b comme collection, et index comme numéro sur lequel on veut intervenir.

Par exemple, pour le champ Name de la collection amComputer.LogicalDrives :

```
PifSetStringVal(amComputer.LogicalDrives(3).Name)
```

Cette syntaxe permet de fixr une valeur pour le nom (Name) du troisième disque logique (LogicalDrive(3)) de la collection.

Remarques

Voir aussi :

- PifSetDoubleVal() [page 175]
- PifSetLongVal() [page 176]
- PifSetNullVal() [page 177]
- PifSetDateVal() [page 174]

PifStrVal()

Syntaxe Basic interne

Function PifStrVal(strID As String) As String

Description

Cette fonction renvoie la chaîne de caractères associée à l'identifiant contenu dans le paramètre **strId**

Entrée

- **strID** : Identifiant de la chaîne de caractères à récupérer.

Sortie

Si l'identifiant n'est pas trouvé, la fonction renvoie une chaîne vide et inscrit une erreur dans le journal de Connect-It. Si l'identifiant est trouvé, la fonction renvoie la chaîne de caractères qui lui est associée.

Exemple

```
If [DeviceType] = "" Then
RetVal = PifStrVal("BRAND_UNKNOWN")
End If
```

PifUserFmtStrToVar()

Syntaxe Basic interne

Function PifUserFmtStrToVar(strData As String, strUserFmtName As String) As Variant

Description

Cette fonction traite une chaîne de caractères en fonction d'un format prédéfini au moyen d'un assistant dans l'interface graphique de Connect-It, et renvoie un nombre variant de type Date ou Nombre en fonction de la nature du format prédéfini.

Entrée

- **strData** : Ce paramètre contient la chaîne à traiter.
- **strUserFmtName** : Ce paramètre contient le nom du format prédéfini.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Si l'on considère les deux formats prédéfinis suivants :

- origine = "yyyy'-'mm'-'dd"
- destination = "'Le 'dddd' 'dd' 'mmmm' 'yyyy"

Alors le script :

```
Dim dTmp as Variant
dTmp=PifUserFmtVarToStr([date_modified],"origine")
RetVal = PifUserFmtStrToVar(dTmp,"destination")
```

Renvoie pour [date_modified]= 2001-05-30 la valeur "Le jeudi 30 mai 2001"

Remarques

Note :

Pour plus d'information sur les formats prédéfinis, consultez le Guide Utilisateur de Connect-It

Voir aussi :

- PifUserFmtVarToStr() [page 184]

PifUserFmtVarToStr()

Syntaxe Basic interne

Function PifUserFmtVarToStr(vData As Variant, strUserFmtName As String) As String

Description

Cette fonction traite un variant en fonction d'un format prédéfini et renvoie une chaîne de caractères.

Entrée

- **vData** : Ce paramètre contient le variant traité par la fonction.
- **strUserFmtName** : Ce paramètre contient le nom du format prédéfini.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Si l'on considère les deux formats prédéfinis suivants :

- origine = "yyyy'-'mm'-'dd"
- destination = "'Le 'dddd' 'dd' 'mmmm' 'yyyy"

Alors le script :

```
Dim dTmp as Variant
dTmp=PifUserFmtVarToStr([date_modified],"origine")
RetVal = PifUserFmtStrToVar(dTmp,"destination")
```

Renvoie pour [date_modified]= 2001-05-30 la valeur "Le jeudi 30 mai 2001"

Remarques

 Note :

Pour plus d'information sur les format prédéfinis, consultez le Guide Utilisateur de Connect-It

Voir aussi :

- PifUserFmtStrToVar() [page 183]

PifWriteBlobInFile()

Syntaxe Basic interne

Function PifWriteBlobInFile(strPath As String, strFileName As String) As Long

Description

Cette fonction permet d'écrire dans un fichier le contenu d'un élément binaire (blob). Si le fichier n'existe pas, il est créé par la fonction. Si le fichier existe, il est écrasé.

Entrée

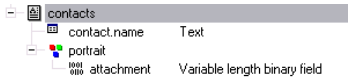
- **strPath** : Ce paramètre contient le chemin complet de l'élément binaire (blob) dans le document source.
- **strFileName** : Ce paramètre contient le nom complet du fichier dans lequel l'objet binaire sera écrit.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

Si l'on considère un document source contenant le nom et la photo d'un employé, comme l'illustre la capture d'écran ci-dessous :



Le script suivant sauvegarde la photo de chaque employé dans le répertoire c:\bitmap. Le nom du fichier est celui de l'employé.

```
Dim lErr As Long
Dim strFileName As String

strFileName = "C:\bitmap\" & ['contact.name'] & ".bmp"
lErr = PifWriteBlobInFile("portrait.attachment", strFileName)
```

Note :

- L'élément **contact.name** contient un point ("."), il est donc nécessaire de l'entourer par des apostrophes pour le référencer (['contact.name'])
- Lorsqu'une fonction renvoie un code d'erreur, il est nécessaire d'affecter une variable au code de retour de la fonction (comme dans le script précédent : **lErr = PifWriteBlobInFile()**). Dans le cas contraire, le script n'est pas valide.

Remarques

La fonction renvoie une erreur dans les cas suivants :

- le chemin du fichier n'est pas valide,

- le chemin de l'élément source n'existe pas dans le document source,
- l'élément source n'est pas de type binaire.

PifXMLDump()

Syntaxe Basic interne

Function PifXMLDump(strPath As String) As String

Description

Cette fonction permet de sauvegarder le contenu d'un élément du document source (et de ses sous-éléments) au format XML, dans une chaîne de caractères.

Entrée

- **strPath** : Ce paramètre contient le chemin de l'élément du document source concerné par l'opération.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

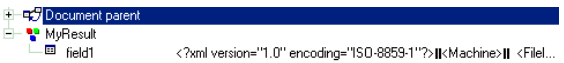
Exemple

Ce premier exemple illustre comment sauvegarder le contenu de l'élément ci-dessous en utilisant la fonction **PifXMLDump()**.

Pour ce faire, on utilise le script ci-dessous, associé au champ "field1"

```
RetVal = PifXMLDump("")
```

La capture d'écran ci-dessous illustre le résultat de l'exécution de la fonction.



Le champ "field1" contient le document XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Machine>
<FileInfo>
<FileName>F:\ConnectIt4016\datakit\idd\Frankfurt01.fsf</FileName>
</FileInfo>
</Machine>
```

Dans ce même exemple, pour sauvegarder uniquement la structure de l'élément FileInfo, on utilisera le script suivant :

```
RetVal = PifXMLDump("FileInfo")
```

Le document XML contenu dans le champ sera alors le suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FileInfo>
<FileName>F:\ConnectIt4016\datakit\idd\Frankfurt01.fsf</FileName>
</FileInfo>
```

PMT()

Syntaxe Basic interne

Function PMT(dblRate As Double, iNper As Long, dblPV As Double, dblFV As Double, iType As Long) As Double

Description

Cette fonction renvoie le montant d'une annuité basée sur des versements constants et périodiques, et sur un taux d'intérêt fixe.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière.
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.

- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les paramètres **Rate** et **Nper** doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités.
- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

PPMT()

Syntaxe Basic interne

Function PPMT(dblRate As Double, iPer As Long, iNper As Long, dblPV As Double, dblFV As Double, iType As Long) As Double

Description

Cette fonction renvoie le montant du remboursement du capital, pour une échéance donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **iPer** : Ce paramètre indique la période concernée par le calcul, comprise entre 1 et la valeur du paramètre **Nper**.
- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.
- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les paramètres **Rate** et **Nper** doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités.
- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

PV()

Syntaxe Basic interne

Function PV(dblRate As Double, iNper As Long, dblPmt As Double, dblFV As Double, iType As Long) As Double

Description

Cette fonction renvoie le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques, et sur un taux d'intérêt fixe.

Entrée

- **dblRate** : Ce paramètre indique le taux d'intérêt par échéance. Par exemple pour un prêt à taux d'intérêt annuel de 6%, remboursé suivant une périodicité mensuelle, le taux par échéance est de :

```
0,06/12=0,005 soit 0,5%
```

- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière.
- **dblPmt** : Ce paramètre indique le montant du paiement à effectuer à chaque échéance. Le paiement comporte généralement le principal et les intérêts.
- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les paramètres **Rate** et **Nper** doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités.
- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.

Randomize()

Syntaxe Basic interne

Function Randomize(lValue As Long)

Description

Initialise le générateur de nombres aléatoires.

Entrée

- **lValue** : Paramètre optionnel utilisé pour initialiser le générateur de nombres aléatoires de la fonction **Rnd** en lui donnant une nouvelle valeur initiale. Si ce paramètre est omis, la valeur renvoyée par l'horloge système est utilisée comme valeur initiale.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyNumber
Randomize
MyNumber= Int((10*Rnd)+1) :'Renvoie une valeur aléatoire comprise entre 1
et 10.
RetVal=MyNumber
```

Remarques

Voir aussi :

- Rnd() [page 201]

RATE()

Syntaxe Basic interne

Function RATE(iNper As Long, dblPmt As Double, dblFV As Double, dblPV As Double, iType As Long, dblGuess As Double) As Double

Description

Cette fonction renvoie le taux d'intérêt par échéance pour une annuité.

Entrée

- **iNper** : Ce paramètre contient le nombre total d'échéances de l'opération financière.
- **dblPmt** : Ce paramètre indique le montant du paiement à effectuer à chaque échéance. Le paiement comporte généralement le principal et les intérêts.
- **dblFV** : Ce paramètre contient la valeur future ou le solde que vous souhaitez obtenir après avoir effectué le dernier paiement. En règle générale, et dans le cas d'un remboursement d'un emprunt en particulier, ce paramètre prend la valeur "0". En effet, une fois toutes les échéances remboursées, la valeur de l'emprunt est nulle.
- **dblPV** : Ce paramètre contient la valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur.
- **iType** : Ce paramètre indique la date d'échéance des paiements. Il peut prendre les valeurs suivantes :
  - **0** si les paiements sont dus à terme échu (c'est à dire en fin de période)
  - **1** si les paiements sont dus à terme à échoir (c'est à dire en début de période)
- **dblGuess** : Ce paramètre contient la valeur estimée du taux d'intérêt par échéance.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Remarques

Note :

- Les sommes versées (exprimées notamment par le paramètre **Pmt**) sont représentées par des nombres négatifs. Les sommes reçues sont représentées par des nombres positifs.
- Cette fonction effectue les calculs par itération, en commençant par la valeur attribuée au paramètre **Guess**. Si aucun résultat n'est trouvé au bout de vingt itérations, la fonction échoue.

RemoveRows()

Syntaxe Basic interne

Function RemoveRows(strList As String, strRowNames As String) As String

Description

Supprime dans une liste les lignes identifiées par le paramètre **strRowNames**.

Cette fonction est utile lors du traitement des valeurs d'un contrôle de type "ListBox". Les valeurs d'un tel contrôle sont représentées par des chaînes bi-dimensionnelles dont les caractéristiques sont les suivantes :

- Le caractère "|" est utilisé comme séparateur de colonnes.
- Le caractère "," est utilisé comme séparateur de lignes.
- Chaque ligne est terminée par un identifiant unique situé à droite du signe "="

Entrée

- **strList** : Chaîne source contenant les valeurs d'un contrôle "ListBox" à traiter.
- **strRowNames** : Identifiants des lignes à supprimer. Les identifiants sont séparés par des virgules.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=RemoveRows("a1|a2=a0,b1|b2=b0", "a0,c0") :'Renvoie "b1|b2=b0"
RetVal=MyStr
```

Remarques

Voir aussi :

- SubList() [page 212]
- SetSubList() [page 205]
- ApplyNewVals() [page 59]

Replace()

Syntaxe Basic interne

Function Replace(strData As String, strOldPattern As String, strNewPattern As String, bCaseSensitive As Long) As String

Description

Remplace toutes les occurrences du paramètre **strOldPattern** par la valeur du paramètre **strNewPattern** au sein de la chaîne de caractères contenue dans **strData**. La recherche de **strOldPattern** peut tenir compte ou non de la casse en fonction de la valeur du paramètre **bCaseSensitive**.

Entrée

- **strData** : Chaîne de caractères contenant les occurrences à remplacer.
- **strOldPattern** : Occurrence à recherche dans la chaîne de caractères contenue dans **strData**.
- **strNewPattern** : Texte remplaçant toute occurrence trouvée.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse. Par défaut ce paramètre a pour valeur 1.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=Replace("toimoitoimoitoi", "toi", "moi",0) :'Renvoie "moimoimoimoimo
i"
MyStr=Replace("toimoitoimoitoi", "Toi", "moi",1) :'Renvoie "toimoitoimoito
i"
MyStr=Replace("toimoiToimoitoi", "Toi", "moi",1) :'Renvoie "toimoimoimoito
i"
```

Right()

Syntaxe Basic interne

Function Right(strString As String, lNumber As Long) As String

Description

Renvoie iNumber caractères d'une chaîne en partant de la droite.

Entrée

- **strString** : Chaîne de caractères à traiter.
- **lNumber** : Nombre de caractères à renvoyer.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim lWord, strMsg, rWord, iPos :' Declare variables.
strMsg = "Left() Test."
iPos = InStr(1, strMsg, " ") :' Find space.
lWord = Left(strMsg, iPos - 1) :' Get left word.
rWord = Right(strMsg, Len(strMsg) - iPos) :' Get right word.
```

```
strMsg=rWord+lWord :' And swap them
RetVal=strMsg
```

RightPart()

Syntaxe Basic interne

Function RightPart(strFrom As String, strSep As String, bCaseSensitive As Long) As String

Description

Extrait la portion d'une chaîne de caractères située à droite du séparateur précisé dans le paramètre **strSep**.

La recherche du séparateur s'effectue de la droite vers la gauche.

La recherche peut tenir compte ou non de la casse en fonction de la valeur du paramètre **bCaseSensitive**.

Entrée

- **strFrom** : Chaîne source à traiter.
- **strSep** : Caractère utilisé comme séparateur dans la chaîne source.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Ces exemples illustrent l'utilisation des fonctions **LeftPart**, **LeftPartFromRight**, **RightPart** et **RightPartFromLeft** sur une même chaîne de caractères: "Ceci_est_un_test" :

```
LeftPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci".

```
LeftPartFromRight("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci_est_un".

```
RightPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "test".

```
RightPartFromLeft("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "est_un_test".

RightPartFromLeft()

Syntaxe Basic interne

Function RightPartFromLeft(strFrom As String, strSep As String, bCaseSensitive As Long) As String

Description

Extrait la portion d'une chaîne de caractères située à droite du séparateur précisé dans le paramètre **strSep**.

La recherche du séparateur s'effectue de la gauche vers la droite.

La recherche peut tenir compte ou non de la casse en fonction de la valeur du paramètre **bCaseSensitive**.

Entrée

- **strFrom** : Chaîne source à traiter.
- **strSep** : Caractère utilisé comme séparateur dans la chaîne source.
- **bCaseSensitive** : En fonction de la valeur de ce paramètre, la recherche respecte (=1) ou non (=0) la casse. Par défaut ce paramètre a pour valeur 1.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Ces exemples illustrent l'utilisation des fonctions **LeftPart**, **LeftPartFromRight**, **RightPart** et **RightPartFromLeft** sur une même chaîne de caractères: "Ceci_est_un_test" :

```
LeftPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci".

```
LeftPartFromRight("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "Ceci_est_un".

```
RightPart("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "test".

```
RightPartFromLeft("Ceci_est_un_test","_",0)
```

Renvoie la chaîne "est_un_test".

RmAllInDir()

Syntaxe Basic interne

Function RmAllInDir(strRmDirectory As String, bStopIfError As Long) As Long

Description

Cette fonction efface tous les éléments (fichiers et dossiers) d'un dossier. Le dossier lui-même n'est pas supprimé.

Entrée

- **strRmDirectory** : Ce paramètre contient le chemin complet du dossier concerné par l'opération.
- **bStopIfError** : Si ce paramètre vaut 1, l'opération d'effacement est interrompue si la suppression d'un fichier ou d'un dossier échoue. Si ce paramètre vaut 0, l'opération d'effacement est poursuivie et passe au fichier ou au dossier suivant.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
RetVal = RmAllInDir("c:\files\test", 1)
```

RmDir()

Syntaxe Basic interne

Function RmDir(strRmDirectory As String) As Long

Description

Détruit un répertoire.

Entrée

- **strRmDirectory** : Chemin complet du répertoire à détruire.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Exemple

```
RetVal = RmDir("c: mp")
```

Remarques

Note :

Le répertoire à détruire doit être vide. Dans le cas contraire, la fonction est inopérante.

Rnd()

Syntaxe Basic interne

Function Rnd(dValue As Double) As Double

Description

Renvoie une valeur contenant un nombre aléatoire.

Entrée

- **dValue** : Paramètre optionnel dont la valeur définit le mode de génération adopté par la fonction :
  - Inférieur à zéro : Le même nombre est généré à chaque fois.
  - Supérieur à zéro : Nombre aléatoire suivant dans la série.
  - Egal à zéro : Dernier nombre aléatoire généré.
  - Omis : Nombre aléatoire suivant dans la série.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyNumber
Randomize
MyNumber= Int((10*Rnd)+1) :'Renvoie une valeur aléatoire comprise entre 1
et 10.
RetVal=MyNumber
```

Remarques

Note :

Avant d'appeler cette fonction, vous devez utiliser la fonction **Randomize**, sans aucun paramètre, pour initialiser le générateur de nombres aléatoires.

Voir aussi :

- Randomize() [page 192]

RoundValue()

Syntaxe Basic interne

Function RoundValue(dValue As Double, iDigits As Long) As Double

Description

Cette fonction calcule l'arrondi d'un nombre au nombre de chiffres après la virgule précisé par le paramètre **iDigits**.

Entrée

- **dValue** : Ce paramètre contient le nombre à arrondir.
- **iDigits** : Ce paramètre contient le nombre de chiffres après la virgule à conserver pour l'arrondi.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple ci-dessous :

```
RetVal = RoundValue(1.2568, 2)
```

renvoie la valeur :

```
1.26
```

L'exemple ci-dessous :

```
RetVal = RoundValue(1.2568, 0)
```

renvoie la valeur :

```
1
```

RTrim()

Syntaxe Basic interne

Function RTrim(strString As String) As String

Description

Supprime tous les espaces suivant le dernier caractère (qui n'est pas un espace) d'une chaîne.

Entrée

- **strString** : Chaîne de caractères à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
' This example uses the LTrim and RTrim functions to strip leading ' and t
railing spaces, respectively, from a string variable.
' It uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Dim strString as String
Dim strTrimString as String
strString = "  <-Trim->  " :' Initialize string.
strTrimString = LTrim(strString) :' strTrimString = "<-Trim->  ".
strTrimString = LCase(RTrim(strString)) :' strTrimString = "  <-trim->".
strTrimString = LTrim(RTrim(strString)) :' strTrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
strTrimString = UCase(Trim(strString)) :' strTrimString = "<-TRIM->".
RetVal= "|" & strTrimString & "|"
```

Second()

Syntaxe Basic interne

Function Second(tmTime As Date) As Long

Description

Renvoie le nombre de secondes contenu dans la l'heure exprimée par le paramètre **tmTime**.

Entrée

- **tmTime** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strSecond
strSecond=Second(Date())
RetVal=strSecond :'Renvoie le nombre de secondes écoulées dans l'heure cou
rante par exemple "30" s'il est actuellement 15:45:30
```

SetMaxInst()

Syntaxe Basic interne

Function SetMaxInst(lMaxInst As Long) As Long

Description

Cette fonction permet de fixer le nombre maximal d'instructions qu'un script Basic peut exécuter. Par défaut, le nombre d'instructions est limité à 10000.

Entrée

- **lMaxInst** : Ce paramètre contient le nombre d'instructions maximal exécutables par un script.

Sortie

- 0 : La fonction s'est exécutée normalement.
- Non nul : Code d'erreur.

Remarques

 Note :

Si vous affectez la valeur "0" au paramètre **lMaxInst**, le nombre d'instructions exécutables par un script est illimité.

SetSubList()

Syntaxe Basic interne

Function SetSubList(strValues As String, strRows As String, strRowFormat As String) As String

Description

Définit les valeurs d'une sous-liste pour un contrôle "ListBox".

Entrée

- **strValues** : Chaîne source contenant les valeurs d'un contrôle "ListBox" à traiter.
- **strRows** : Liste de valeurs à ajouter ou à substituer à celles de la chaîne contenue dans le paramètre **strValues**. Les valeurs sont séparées par le caractère "|". Les lignes traitées sont identifiées par leur identifiant, situé à droite du signe "=". Les lignes inconnues ne sont pas traitées.

- **strRowFormat** : Instructions de formatage de la sous-liste. Les instructions sont séparées par le caractère "|". Ce paramètre possède les caractéristiques suivantes :
  - "1" représente les informations contenues dans la première colonne de la sous-liste.
  - "i-j" peut être utilisé pour définir un ensemble de colonnes.
  - "-" prend en compte toutes les colonnes.
  - Une colonne inconnue ne renvoie aucune valeur.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=SetSubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "A2|A1=a0, B2|B1=b
0", "2|1") :'Renvoie "A1|A2|a3=a0,B1|B2|b3=b0,c1|c2|c3=c0"
MyStr=SetSubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "Z2=*,B2=b0", "2")
:'Renvoie "a1|Z2|a3=a0,b1|B2|b3=b0,c1|Z2|c3=c0"
MyStr=SetSubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "B5|B6|B7=b0,C5|C6
,C7=c0", "5-7") :'Renvoie "a1|a2|a3=a0,b1|b2|b3||B5|B6|B7=b0,c1|c2|c3||C5|
C6|C7=c0"
MyStr=SetSubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "B1|B2|B3|B4=b0",
"-") :'Renvoie "a1|a2|a3=a0,B1|B2|B3|B4=b0,c1|c2|c3=c0"
MyStr=SetSubList("A|B|C,D|E|F", "X=*", "2") :'Renvoie "A|X|C,D|X|F"
RetVal=""
```

Sgn()

Syntaxe Basic interne

Function Sgn(dValue As Double) As Double

Description

Renvoie une valeur indiquant le signe d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître le signe.

Sortie

La fonction peut renvoyer une des valeurs suivante :

- 1 : Le nombre est supérieur à zéro.
- 0 : Le nombre est égal à zéro.
- -1 : Le nombre est inférieur à zéro.

Exemple

```
Dim dNumber as Double
dNumber=-256
RetVal=Sgn(dNumber)
```

Shell()

Syntaxe Basic interne

Function Shell(strExec As String, bShowWindow As Long, bBackground As Long) As Long

Description

Lance un programme exécutable.

Entrée

- **strExec** : Chemin complet de l'exécutable à lancer.
- **bShowWindow** : Si ce paramètre a pour valeur 1 (valeur par défaut), la boîte de commande s'affiche au lancement du programme. Si ce paramètre a pour valeur 0, la boîte de commande ne s'affiche pas.
- **bBackground** : Si ce paramètre a pour valeur 1 (valeur par défaut), la fonction attend la fin de l'exécution du programme pour rendre la main (exécution synchrone). Si ce paramètre a pour valeur 0, l'exécution du programme est asynchrone.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyId
MyId=Shell("C:\WinNT\explorer.exe")
RetVal=""
```

Sin()

Syntaxe Basic interne

Function Sin(dValue As Double) As Double

Description

Renvoie le sinus d'un nombre, exprimé en radians.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître le sinus.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dCalc as Double
dCalc=Sin(2.79)
RetVal=dCalc
```

Remarques

Note :

La formule de conversion des degrés en radians est la suivante :

```
angle en radians = (angle en degrés) * Pi / 180
```

Space()

Syntaxe Basic interne

Function Space(iCount As Long) As String

Description

Crée une chaînes de caractères comprenant le nombre d'espaces indiqué par le paramètre **iSpace**.

Entrée

- **iCount** : Nombre d'espaces à insérer dans la chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyString
' Renvoie une chaîne de 10 espaces.
MyString = Space(10)
:' Insère 10 espaces entre deux chaînes.
MyString = "Espace" & Space(10) & "inséré"
RetVal=MyString
```

Remarques

Note :

Cette fonction peut servir à formater des chaînes ou à effacer des données dans des chaînes de longueur fixe.

Sqr()

Syntaxe Basic interne

Function Sqr(dValue As Double) As Double

Description

Renvoie la racine carrée d'un nombre.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la racine carrée.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dCalc as Double
dCalc=Sqr(81)
RetVal=dCalc
```

Str()

Syntaxe Basic interne

Function Str(strValue As String) As String

Description

Convertit un nombre en une chaîne de caractères.

Entrée

- **strValue** : nombre à convertir en chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dNumber as Double
dNumber=Cos(2.79)
RetVal=Str(dCalc)
```

StrComp()

Syntaxe Basic interne

Function StrComp(strString1 As String, strString2 As String, iOptionCompare As Long) As Long

Description

Effectue la comparaison entre deux chaînes de caractères.

Entrée

- **strString1** : Première chaîne de caractères.
- **strString2** : Deuxième chaîne de caractères.
- **iOptionCompare** : type de comparaison. Ce paramètre peut prendre la valeur "0" pour une comparaison binaire, ou "1" pour une comparaison du texte des deux chaînes.

Sortie

- -1 : **strString1** est supérieure à **strString2**.
- 0 : **strString1** est égale à **strString2**.
- 1 : **strString1** est inférieure à **strString2**.

String()

Syntaxe Basic interne

Function String(iCount As Long, strString As String) As String

Description

Renvoie une chaîne composée de **iCount** fois le caractère **strString**.

Entrée

- **iCount** : Nombre d'occurrences du caractère **strString**.
- **strString** : caractère utilisé pour la composition de la chaîne.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim iCount as Integer
Dim strTest as String
strTest="T"
iCount=5
RetVal=String(iCount,strTest)
```

SubList()

Syntaxe Basic interne

Function SubList(strValues As String, strRows As String, strRowFormat As String) As String

Description

Renvoie une sous-liste d'une liste de valeurs contenue dans une chaîne de caractères représentant les valeurs d'un contrôle "ListBox".

Entrée

- **strValues** : Chaîne source contenant les valeurs d'un contrôle "ListBox" à traiter.
- **strRows** : Identifiants des lignes à inclure dans la sous-liste. Les identifiants sont séparés par une virgule. Certains jokers sont acceptés :
  - "*" inclut tous les identifiants dans la sous-liste.
  - Un identifiant inconnu renvoie une valeur vide pour la sous-liste.
- **strRowFormat** : Instructions de formatage de la sous-liste. Les instructions sont séparées par le caractère "|". Ce paramètre possède les caractéristiques suivantes :
  - "1" représente les informations contenues dans la première colonne de la liste dont on extrait une sous-liste.
  - "0" représente l'identifiant de la ligne de la liste dont on extrait une sous-liste.
  - "*" représente les informations contenues dans toutes les colonnes (à l'exception de l'identifiant de la ligne).
  - Une colonne inconnue ne renvoie aucune valeur.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "a0,b0,a0", "3|2|3")
:'Renvoie "a3|a2|a3,b3|b2|b3,a3|a2|a3"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "*|0") :'Renvoie
"a1|a2|a3|a0,b1|b2|b3|b0,c1|c2|c3|c0"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "*=0") :'Renvoie
"a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "999=0") :'Renvo
ie "=a0,=b0,=c0"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "z0", "*=0") :'Renvoi
e ""
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "=1") :'Renvoie
"=a1,=b1,=c1"
MyStr=SubList("A|B|C,D|E|F", "*", "2=0") :'Renvoie "B,E"
RetVal=""
```

Tan()

Syntaxe Basic interne

Function Tan(dValue As Double) As Double

Description

Renvoie la tangente d'un nombre, exprimé en radians.

Entrée

- **dValue** : Nombre dont vous souhaitez connaître la tangente.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim dCalc as Double
dCalc=Tan(2.79)
RetVal=dCalc
```

Remarques

Note :

La formule de conversion des degrés en radians est la suivante :

```
angle en radians = (angle en degrés) * Pi / 180
```

Time()

Syntaxe Basic interne

Function Time() As Date

Description

Renvoie l'heure courante.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
RetVal = Time()
```

Timer()

Syntaxe Basic interne

Function Timer() As Double

Description

Renvoie le nombre de secondes écoulées depuis 12:00 AM.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
RetVal = Timer()
```

TimeSerial()

Syntaxe Basic interne

Function TimeSerial(iHour As Long, iMinute As Long, iSecond As Long) As Date

Description

Cette fonction renvoie une heure formatée en fonction des paramètres **iHour**, **iMinute** et **iSecond**.

Entrée

- **iHour** : Heure.
- **iMinute** : Minutes.
- **iSecond** : Secondes.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

Chacun de ces paramètres peut prendre pour valeur une expression numérique représentant un nombre d'heures, de minutes ou de secondes. Ainsi l'exemple suivant :

```
TimeSerial(12-8, -10, 0)
```

renvoie la valeur :

```
3:50:00
```

Lorsque la valeur d'un paramètre est en dehors de l'intervalle de valeurs généralement admis (c'est à dire 0-59 pour les minutes et les secondes et 0-23 pour les heures), elle est convertie vers le paramètre immédiatement supérieur. Ainsi, si vous entrez "75" comme valeur pour le paramètre **iMinute**, ce dernier sera interprété comme 1 heure et 15 minutes.

L'exemple suivant :

```
TimeSerial (16, 50, 45)
```

renvoie la valeur :

```
16:50:45
```

TimeValue()

Syntaxe Basic interne

Function TimeValue(tmTime As Date) As Date

Description

Cette fonction renvoie la partie heure d'une valeur "Date+Heure"

Entrée

- **tmTime** : Date au format "Date+Heure".

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant :

```
TimeValue ("1999/09/24 15:00:00")
```

renvoie la valeur :

```
15:00:00
```

ToSmart()

Syntaxe Basic interne

Function ToSmart(strString As String) As String

Description

Cette fonction reformate un chaîne source en mettant des majuscules au début de chaque mot.

Entrée

- **strString** : Chaîne source à reformater.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple suivant :

```
RetVal = ToSmart ("hello world")
```

renvoie la valeur :

```
Hello World
```

Trim()

Syntaxe Basic interne

Function Trim(strString As String) As String

Description

Supprime tous les espaces précédant le premier caractère (qui n'est pas un espace) d'une chaîne et tous les espaces suivant le dernier caractère (qui n'est pas un espace) d'une chaîne.

Entrée

- **strString** : Chaîne de caractères à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
' This example uses the LTrim and RTrim functions to strip leading ' and t
railing spaces, respectively, from a string variable.
' It uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Dim strString as String
Dim strTrimString as String
strString = "  <-Trim->  " :' Initialize string.
strTrimString = LTrim(strString) :' strTrimString = "<-Trim->  ".
strTrimString = LCase(RTrim(strString)) :' strTrimString = "  <-trim->".
strTrimString = LTrim(RTrim(strString)) :' strTrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
strTrimString = UCase(Trim(strString)) :' strTrimString = "<-TRIM->".
RetVal= "|" & strTrimString & "|"
```

UCase()

Syntaxe Basic interne

Function UCase(strString As String) As String

Description

Passe tous les caractères d'une chaîne en majuscules.

Entrée

- **strString** : Chaîne de caractères à passer en majuscules.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
' This example uses the LTrim and RTrim functions to strip leading ' and t
railing spaces, respectively, from a string variable.
' It uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls
```

```
Dim strString as String
Dim strTrimString as String
strString = "  <-Trim->  " :' Initialize string.
strTrimString = LTrim(strString) :' strTrimString = "<-Trim->  ".
strTrimString = LCase(RTrim(strString)) :' strTrimString = "  <-trim->".
strTrimString = LTrim(RTrim(strString)) :' strTrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
strTrimString = UCase(Trim(strString)) :' strTrimString = "<-TRIM->".
RetVal= "|" & strTrimString & "|"
```

UnEscapeSeparators()

Syntaxe Basic interne

Function UnEscapeSeparators(strSource As String, strEscChar As String) As String

Description

Supprime tous les caractères d'échappement d'une chaîne de caractères.

Entrée

- **strSource** : Chaîne de caractères à traiter.
- **strEscChar** : Caractère d'échappement à supprimer.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=UnEscapeSeparators("toi\|moi\|toi\|", "\") :'Renvoie la valeur "toi|
moi|toi|"
RetVal=""
```

Union()

Syntaxe Basic interne

Function Union(strListOne As String, strListTwo As String, strSeparator As String, strEscChar As String) As String

Description

Rassemble deux chaînes de caractères délimitées par des séparateurs. Les doublons sont supprimés.

Entrée

- **strListOne** : Première chaîne de caractères.
- **strListTwo** : Deuxième chaîne de caractères.
- **strSeparator** : Séparateur utilisé pour délimiter les éléments contenus dans les chaînes.
- **strEscChar** : Caractère d'échappement. Si ce caractère préfixe le séparateur, ce dernier est ignoré.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim MyStr
MyStr=Union("a1|a2,b1|b2", "a1|a3,b1|b2", ",", "\") :'Renvoie la valeur "a
1|a2,b1|b2,a1|a3"
MyStr=Union("a1|a2,b1|b2", "a1|a3\,b1|b2", ",", "\") :'Renvoie la valeur "
a1|a2,b1|b2,a1|a3\,b1|b2"
RetVal=""
```

UTCToLocalDate()

Syntaxe Basic interne

Function UTCToLocalDate(tmUTC As Date) As Date

Description

Cette fonction convertit une date au format UTC (indépendante d'un quelconque fuseau horaire) en une date au format "Date+Heure".

Entrée

- **tmUTC** : Date au format UTC.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
RetVal = UTCToLocaldate([DateTime])
```

Val()

Syntaxe Basic interne

Function Val(strString As String) As Double

Description

Convertit une chaîne de caractères représentant un nombre en un nombre de type "Double".

Entrée

- **strString** : Chaîne à convertir.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strYear
Dim dYear as Double
strYear=Year(Date())
dYear=Val(strYear)
RetVal=dYear :'Renvoie l'année en cours
```

WeekDay()

Syntaxe Basic interne

Function WeekDay(tmDate As Date) As Long

Description

Renvoie le jour de la semaine contenu dans la date exprimée par le paramètre **tmDate**.

Entrée

- **tmDate** : Paramètre au format Date+Heure à traiter.

Sortie

Le nombre retourné correspond à un jour de la semaine, le "1" représentant le dimanche, le "2" le lundi, ..., le "7" le samedi.

Exemple

```
Dim strWeekDay
strWeekDay=WeekDay(Date())
RetVal=strWeekDay :'Renvoie le jour de la semaine
```

XmlAttribute()

Syntaxe Basic interne

Function XmlAttribute(strName As String, strValue As String) As String

Description

Cette fonction génère la chaîne de texte strName="strValue", où strName est conservé tel quel et les entités prédéfinies du langage XML contenues dans strValue sont converties en langage XML.

Les cinq entités prédéfinies en XML et leurs interprétations respectives sont :

- l'entité < qui correspond au caractère <
- l'entité > qui correspond au caractère >
- l'entité & qui correspond au caractère &
- l'entité ' qui correspond au caractère '
- l'entité " qui correspond au caractère "

Entrée

- **strName** : Ce paramètre contient le nom de l'attribut XML.
- **strValue** : Ce paramètre contient la valeur de l'attribut XML.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

L'exemple ci-dessous :

```
RetVal = XmlAttribute("Equation & condition","dix < onze")
```

renvoie la valeur :

```
Equation & condition = "dix &lt; onze"
```

Year()

Syntaxe Basic interne

Function Year(tmDate As Date) As Long

Description

Renvoie l'année contenue dans la date exprimée par le paramètre **tmDate**.

Entrée

- **tmDate** : Paramètre au format Date+Heure à traiter.

Sortie

En cas d'erreur, un message est inscrit dans le fichier journal de Connect-It.

Exemple

```
Dim strYear
strYear=Year(Date())
RetVal=strYear :'Renvoie l'année en cours
```

III Index

Fonctions disponibles - Liste complète des fonctions

- **Abs**
- **AppendOperand**
- **ApplyNewVals**
- **Asc**
- **Atn**
- **BasicToLocalDate**
- **BasicToLocalTime**
- **BasicToLocalTimeStamp**
- **Beep**
- **CDbl**
- **ChDir**
- **ChDrive**
- **Chr**
- **CInt**
- **CLng**
- **Cos**
- **CountOccurences**
- **CountValues**
- **CSng**
- **CStr**
- **CurDir**
- **CVar**
- **Date**
- **DateAdd**
- **DateAddLogical**
- **DateDiff**
- **DateSerial**
- **DateValue**
- **Day**
- **EscapeSeparators**
- **ExeDir**
- **Exp**
- **ExtractValue**
- **FileCopy**
- **FileDateTime**
- **FileExists**
- **FileLen**
- **Fix**
- **FormatDate**
- **FormatResString**

- FV
- GetEnvVar
- GetListItem
- GetXmlAttributeValue
- GetXmlElementValue
- Hex
- Hour
- InStr
- Int
- IPMT
- IsNumeric
- Kill
- LCase
- Left
- LeftPart
- LeftPartFromRight
- Len
- LocalToBasicDate
- LocalToBasicTime
- LocalToBasicTimeStamp
- LocalToUTCDate
- Log
- LTrim
- MakeInvertBool
- Mid
- Minute
- MkDir
- Month
- Name
- Now
- NPER
- Oct
- ParseDate
- ParseDMYDate
- ParseMDYDate
- ParseYMDDate
- PifCloseODBCDatabase
- PifCreateDynaMaptableFromFmtName
- PifDateToTimezone
- PifExecODBCSql
- PifFirstInCol
- PifGetBlobSize
- PifGetDateVal
- PifGetDoubleVal
- PifGetElementChildName
- PifGetElementCount
- PifGetHexStringFromBlob
- PifGetInstance
- PifGetIntlStringVariantVal
- PifGetIntVal
- PifGetItemCount
- PifGetLongVal
- PifGetOpenSessionTime
- PifGetParamValue
- PifGetStringFromBlob
- PifGetStringVal
- PifGetVariantVal
- PifIgnoreCollectionMapping
- PifIgnoreDocumentMapping
- PifIgnoreDocumentReconc
- PifIgnoreNodeMapping
- PifIgnoreNodeReconc
- PifIgnoreSubDocumentReconc
- PifIsInMap
- PifLogInfoMsg
- PifLogWarningMsg
- PifMapValue
- PifMapValueContaining
- PifMapValueContainingEx
- PifMapValueEx

- PifNewQueryFromFmtName
- PifNewQueryFromXml
- PifNodeExists
- PifOpenODBCDatabase
- PifQueryClose
- PifQueryGetDateVal
- PifQueryGetDoubleVal
- PifQueryGetIntVal
- PifQueryGetLongVal
- PifQueryGetStringVal
- PifQueryNext
- PifRejectCollectionMapping
- PifRejectDocumentMapping
- PifRejectDocumentReconc
- PifRejectNodeMapping
- PifRejectNodeReconc
- PifRejectSubDocumentReconc
- PifScenarioPath
- PifSetDateVal
- PifSetDoubleVal
- PifSetLongVal
- PifSetNullVal
- PifSetParamValue
- PifSetPendingDocument
- PifSetStringVal
- PifStrVal
- PifUserFmtStrToVar
- PifUserFmtVarToStr
- PifWriteBlobInFile
- PifXMLDump
- PMT
- PPMT
- PV
- Randomize
- RATE
- RemoveRows
- Replace
- Right
- RightPart
- RightPartFromLeft
- RmAllInDir
- RmDir
- Rnd
- RoundValue
- RTrim
- Second
- SetMaxInst
- SetSubList
- Sgn
- Shell
- Sin
- Space
- Sqr
- Str
- StrComp
- String
- SubList
- Tan
- Time
- Timer
- TimeSerial
- TimeValue
- ToSmart
- Trim
- UCase
- UnEscapeSeparators
- Union
- UTCToLocalDate
- Val
- WeekDay

Fonctions disponibles - Connect-It

- **GetXmlAttributeValue**
- **GetXmlElementValue**
- **PifCloseODBCDatabase**
- **PifCreateDynaMaptableFromFmtName**
- **PifDateToTimezone**
- **PifExecODBCSql**
- **PifFirstInCol**
- **PifGetBlobSize**
- **PifGetDateVal**
- **PifGetDoubleVal**
- **PifGetElementChildName**
- **PifGetElementCount**
- **PifGetHexStringFromBlob**
- **PifGetInstance**
- **PifGetIntlStringVariantVal**
- **PifGetIntVal**
- **PifGetItemCount**
- **PifGetLongVal**
- **PifGetOpenSessionTime**
- **PifGetStringFromBlob**
- **PifGetStringVal**
- **PifGetVariantVal**
- **PifIgnoreCollectionMapping**
- **PifIgnoreDocumentMapping**
- **PifIgnoreDocumentReconc**
- **PifIgnoreNodeMapping**
- **PifIgnoreNodeReconc**
- **PifIgnoreSubDocumentReconc**
- **PifIsInMap**
- **PifLogInfoMsg**
- **PifLogWarningMsg**
- **PifMapValue**
- **PifMapValueContaining**
- **PifMapValueContainingEx**
- **PifMapValueEx**
- **PifNewQueryFromFmtName**
- **PifNewQueryFromXml**
- **PifNodeExists**
- **PifOpenODBCDatabase**
- **PifQueryClose**

Fonctions disponibles - Builtin

- **Abs**
- **AppendOperand**
- **ApplyNewVals**
- **Asc**
- **Atn**
- **BasicToLocalDate**
- **BasicToLocalTime**
- **BasicToLocalTimeStamp**
- **Beep**
- **CDbl**
- **ChDir**
- **ChDrive**
- **Chr**
- **CInt**
- **CLng**
- **Cos**
- **CountOccurences**
- **CountValues**
- **CSng**
- **CStr**
- **CurDir**
- **CVar**
- **Date**
- **DateAdd**
- **DateAddLogical**
- **DateDiff**
- **DateSerial**
- **DateValue**
- **Day**
- **EscapeSeparators**
- **ExeDir**
- **Exp**
- **ExtractValue**
- **FileCopy**
- **FileDateTime**
- **FileExists**
- **FileLen**
- **Fix**
- **FormatDate**
- **FormatResString**

- FV
- GetEnvVar
- GetListItem
- Hex
- Hour
- InStr
- Int
- IPMT
- IsNumeric
- Kill
- LCase
- Left
- LeftPart
- LeftPartFromRight
- Len
- LocalToBasicDate
- LocalToBasicTime
- LocalToBasicTimeStamp
- LocalToUTCDate
- Log
- LTrim
- MakeInvertBool
- Mid
- Minute
- MkDir
- Month
- Name
- Now
- NPER
- Oct
- ParseDate
- ParseDMYDate
- ParseMDYDate
- ParseYMDDate
- PMT
- PPMT
- PV
- Randomize
- RATE
- RemoveRows
- Replace
- Right
- RightPart
- RightPartFromLeft
- RmAllInDir
- RmDir
- Rnd
- RoundValue
- RTrim
- Second
- SetMaxInst
- SetSubList
- Sgn
- Shell
- Sin
- Space
- Sqr
- Str
- StrComp
- String
- SubList
- Tan
- Time
- Timer
- TimeSerial
- TimeValue
- ToSmart
- Trim
- UCase
- UnEscapeSeparators