HP Connect-It

Software version: 3.91

Programmer's Reference

Document Release Date: 16 February 2009
Software Release Date: February 2009

Legal Notices

Copyright Notices

Restricted Rights Legend

Confidential computer software.

Valid license from HP required for possession, use or copying.

Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services.

Nothing herein should be construed as constituting an additional warranty.

HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Brands

- Adobe®, Adobe logo®, Acrobat® and Acrobat Logo® are trademarks of Adobe Systems Incorporated.
- Corel® and Corel logo® are trademarks or registered trademarks of Corel Corporation or Corel Corporation Limited.
- Java™ is a US trademark of Sun Microsystems, Inc.
- Microsoft®, Windows®, Windows NT®, Windows® XP, Windows Mobile® and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.
- Oracle® is a registered trademark of Oracle Corporation and/or its affiliates.
- UNIX® is a registered trademark of The Open Group.

Table of Contents

I Introduction

1 Programming fundamentals

This chapter presents the fundamentals of programming using the Basic language available in Connect-It. If you already experience in programming and have used other languages, most of the information presented in this chapter will be familiar to you. However, we do recommend reading throught this chapter because certain classic functions have been volontarily left out of or limited in the implementation of Basic in Connect-It.

Introduction to variables

Variables are used to store data during the execution of a program. They are identified by:

- Their name, used to reference the value contained by the variable.
- Their type, which determines which data can be stored in the variable.

In general, a distinction is made between two types of variables:

- Arrays,
- Scalar variables, which include all variables that are not arrays.

Declaring a variable

Variables must be explicitly declared before being used. The syntax of the declaration is as follows:

```
Dim <Name of the variable> [As <Type of the variable>]
```

Note:

The expicit declaration of variables in Connect-It Basic is the same as using the *Option Explicit* keyword in Microsoft Visual Basic.

Variable names must meet the following constraints:

- Start with an uppercase or lowercase letter,
- Must have no more than 40 characters,
- Can contain the letters A to Z and a to z, the numbers 0 to 9, and the underscore character ("_").

  Note:

  Accented characters are authorized but are advised against.

- Reserved keywords may not be used. For example, names of Basic functions and clauses are reserved keywords.

The optional *As* clause enables you to define the type of the defined variable. The type specifies the type of information stored in the variable. The available data types include: *String*, *Integer*, *Variant*, ...

If the *As* clause is omitted, the variable is considered as a *Variant* type.

Single declaration

In the case of a single declaration, each declaration statement concerns a single variable, as shown in the following example:

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

Combined declaration

In the case of a combined declaration, each declaration statement may concern any number of variables, as shown in the following example:

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

Note:

As already described, when the type of the variable is not specified, by default it is considered as a *Variant*. Thus, in the second line of the above example, the type of the variables *A* and *B* is *Variant* and *C* is an *Integer*.

Data types

The following table summarizes the various types available for a function or a parameter:

Type	Description
Integer	Integer from -32,768 to +32,767.
Long	Integer from -2,147,483,647 to +2,147,483,646.
Single	4-bype floating-point number (single precision).
Double	8-byte floating-point number (double precision).
String	Text in which all characters are allowed.
Date	Date or Date+Time.
Variant	Generic type that can represent any type.

Numerical types

The Basic language available in Connect-It offers several numerical types: Integer, Long, Single and Double. Numerical data types usually use less memory than a *Variant*.

If you are sure a variable will systematically store integers (such as 123) and not fractions (such as 3.14), it is better to declare it as an *Integer* or a *Long*. Operations performed on these data types are faster and required less memory than other data types. These data types are particularly well suited to counters used in loops.

If a variable must contain a fractional number, declare is as a *Single* or *Double*.

Note:

Floating point numbers (*Single* or *Double*) can be subject to rounding errors.

The String type

If you are sure a variable will only store a character string, declare it as *String*:

```
Dim MyString As String
```

You will then be able to store character strings in this variable and manipulate its contents using the dedicated character string processing functions:

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

By default, a *String* type variable is of variable size. The allotted size used to store character strings changes according to the size of the data assigned to the variable. However, it is possible to declare a *String* type variable using the following syntax:

```
Dim <Name of the variable> As String * <Size of the stored string>
```

The following example declares a variable containing 20 characters:

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

If you use this variable to store a string of less than 20 characters, spaces will be added to the end of the string as padding up until the intended size. On the other hand, if you store a string over 20 characters, the string will be truncated from the 21st character.

The Variant type

The *Variant* type is a generic type that can substitute for all other types. You do not need to worry about conversion issues between the different data types and *Variant*. Conversion is performed automatically, as shown in the following example:

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

Even though conversion is automatic, make sure you follow the following rules:

- If you perform arithmetic operations on a *Variant*, it must contain a number, even if it is represented by a character string.
- If a concatenation operation involves a *Variant*, use the & operator rather than the + operator.

A *Variant* can also contain two special values: The empty value and the *Null* value.

The empty value

Before a value is assigned for the first time to a *Variant*, it contains the empty value. This value is a particular value and is not the same as 0, an empty string or the *Null* value. To test whether a *Variant* contains the empty value, use the Basic function **IsEmpty()**, as shown in the following example:

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

A *Variant* containing the empty value can be used in expressions. Depending on the situation, it will be processed as the value 0 or an empty string. To reassign the empty value to a *Variant*, use the keyword *Empty*, as shown in the following example:

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

The Null value

The *Null* value is often used in databases to specify missing or unknown values. This value has special qualities:

- Expression that include the *Null* value always return the *Null* value. The *Null* value is said to be propagated in the expressions. If part of the expression is *Null*, then all of the expression is *Null*.
- As a general rule, if a function parameter is set to *Null*, the function returns the *Null* value.

Data arrays

An array enables you to store and reference a set of variables under a single name and use a number (an index) to uniquely identify them. All array items must have the same data type. You cannot create an array containing both *String* and *Double* values. The *Variant* type can be used to work around this limitation.

Declaring an array

An array is a set of variables.

By convention, the following notions are presented as follows:

- Lower limit of the array: Index of the first item.

   Note:

  By default, the lower limit of an array is 0.

- Upper limit of the array: Index of the last item.

   Note:

  The upper limit of an array may not exceed the size of a *Long* (2 147 483 646 items).

Declaring an array is similar to declaring a variable:

```
Dim <Name of the array>(<Upper limit of the array>) [As <Data type of the
variables contained in the array>]
```

Examples:

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

You can also specify the lower limit of the array by using the following declaration:

```
Dim <Name of the array>(<Lower limit of the array> To <Upper limit of the
array>) [As <Data type of the variables contained in the array>]
```

Examples:

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

Limitations

The following limitations apply to arrays in Connect-It Basic:

- Variable size arrays are not supported. In particular, it is not possible to resize an array on the fly.
- Multi-dimensional arrays are not supported.

Control structures

As their name suggests, control structures make it possible to control the execution of a program. There are two sorts of control structures:

- Decision structures: redirect and guide a program as a result of certain conditions,
- Loop structures: make it possible to repeat program sections depending on certain conditions.

Decision structures

A decision structure conditionally executes instructions depending on the results of a test. The following decision structures are available:

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

If...Then

Use this structure to conditionally execute one or more instructions. The syntax of this structure allows for single line and multiple line statements. Single line statements may only execute one single instruction:

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

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

The condition is generally a comparison, however any expression that results in a numerical value can be used. This value is interpreted as **True** or **False** by the Basic code. **False** corresponds to 0; All other values are considered **True**.

If the condition is evaluated as **True**, the instruction or instructions following the keyword **Then** will be executed.

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

Use this structure to define multiple conditional instruction blocks. Only the first of these blocks evaluated as **True** will be executed.

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

The first condition is tested, if the result is evaluated as **False**, the second condition is tested and so on until one of them is evaluated as **True**. The instruction set after the keyword **Then** is executed.

The keyword **Else** is optional. It makes it possible to define an instruction set to be executed if all the conditions are evaluated as **False**.

Note:

You can nest as many **ElseIf** instructions as you like in the decision structure. However, if you systematically compare the same expression with a different value, the syntax of the decision structure can become unnecessarily complex and difficult to read. In this case, we advise you to use a **Select...Case** type decision structure.

Select...Case

This structure serves the same purpose as the previous decision structures, but in general, the resulting code is more readable. A **Select...Case** function performs a single test at the start of the structure and compares the test result with the values given by each **Case** in the structure. If there is a match, the instruction set associated with the **Case** is executed.

```
Select Case <Test>
[Case <value 1>
<Instructions1>]
[Case <value 2>
<Instructions2>]
...
[Case Else
```

```
<Instructionsn>]
End Select
```

The instruction set associated with the **Case Else** keyword is executed if no match is found for the **Case** keywords.

 Important:

Multiple selection is not supported when using the Case instruction.

The following script will execute correctly:

```
Dim i as integer
For i= 1 to 10
Select case i
Case 1 Print ("1")
Case 2 Print ("2")
Case 3 Print ("3")
Case Else Print ("Others")
End Select
Next i
```

The following script produces an error:

```
Dim i as integer
For i= 1 to 10
Select case i
Case 1,2,3 Print ("a lot")
Case 1 to 5 Print ("a lot too")
Case Else Print ("Others")
End Select
Next i
```

▸ Select [page 42].

Loop structures

A loop structure enables you to repeat the execution of a series of instructions. The following loop structures are available:

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

Do...Loop

Use this structure to execute a series of instructions an undefined number of times. The loop is exited when a condition is met or is not met. This condition is a value or an expression that is evaluated as **False** (0) or **True** (not 0).

Note:

It is possible to exit the loop by force by using the **Exit Do** keyword in the executed instructions.

There are several variations on this structure, but the most common one is the following:

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

In this case, the condition is evaluated first. If it is **True**, the instructions are executed and the program returns to the **Do While** keyword, test the condition again and so on. The loop is exited when the condition is evaluated as **False**.

The following example tests the value of a counter, incremented at each iteration of the loop. The loop is executed when the counter reaches 20.

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

The following example is based on the previous one but exits the loop by force using the **Exit Do** keyword if the counter contains the value 10.

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

In this type of **Do...Loop** structure, the condition is evaluated before executing the instructions. If you wish to execute the instruction and then test the condition, use the following **Do...Loop** structure:

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

Note:

This type of structure guarantees that at least one of the instructions will be executed.

The two previous **Do...Loop** structures iterate for as long as the condition is **True**. If you wish to iterate while the condition is **False**, use one of the following structures:

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

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

Using this structure type, the previous example can be written:

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

For...Next

Use this structure to execute a series of instructions an undefined number of times. Unlike **Do...Loop**, a **For...Next** loop uses a variable called a counter whose value is incremented or decremented at each iteration.

Note:

It is possible to exit the loop by force by using the **Exit For** keyword in the executed instructions.

```
For <Counter> = <Initial value> To <Final value> [Step <Increment>]
<Instructions>
Next [<Counter>]
```

Important:

The arguments **Counter**, **Initial value**, **Final value** and **Increment** are all represented by numerical values.

Note:

Increment may be a positive or negative value. If it is positive, the **Initial value** must be less than or equal to the **Final value** in order for the instructions to be executed. If it is negative, the **Initial value** must be greater than or equal to the **Final value** in order for the instructions to be executed. If the **Increment** is not specified, by default it is set to 1.

When a **For...Next** loop is executed, the following operations are performed:

1. The counter initializes and stores the initial value,
2. The Basic codes tests whether the value of the counter is greater than the final value. If this is the case, the program exits the loop.

Note:

If the increment is negative, the Basic test whether the value of the counter is less than the final value.

3 The instructions are executed,

4 The counter incremented by 1 or the specified value,

5 Operations 2 through 4 are repeated.

The following operation sums all even number up to 1000:

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

The following example is based on the previous one but exits the loop by force using the **Exit For** keyword if the counter contains the value 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
```

Operators

Operators are symbols than enable you to perform simple operations (addition, multiplication, etc.) on variables or compare them. There are several different types of counters:

- Assignment operators,
- Arithmetic operators,
- Relational operators (also called assignment operators),
- Logical operators.

Assigment operators

This type of operator enables you to assign a value to a variable. Connect-It Basic uses one single assigment operator, the "=" sign. The assigment syntax is as follows:

```
<Variable> = <Value>
```

Arithmetic operators

Arithmetic operators enable you to modify the value of a variable arithmetically, or to perform simple arithmetic operations between two expressions.

The + operator

This operator enables you to sum two values. The syntax is as follows:

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

Note:

This operator is used both to sum two numbers and to concatenate strings. To avoid any ambiguity, we recommend you use this operator just for sum operations and to use the & operator to concatenate strings.

The - operator

This operator enables you to differentiate between two values or to negatively sign (monadic operator) a value. The operator has two syntaxes:

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

or

```
- <Expression>
```

The * operator

This operator enables you to multiply two values. The syntax is the following:

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

The / operator

This operator enables you to perform a division between two values. The syntax is as follows:

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

The ^ operator

This operator enables you to raise a value to the power of an exponent. The syntax is as follows:

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

> **Note:**
>
> In this syntax, expression 1 cannot be negative if expression 2 (the exponent) is an integer. When an expression performs several exponential operations in a series, they are interpreted logically from left to right.

The Mod operator

This operator calculates the remainder of the euclidien division of two values. The syntax is as follows:

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

> **Note:**
>
> Floating-point numbers are automatically rounded to integers.

The following example returns 4 (6.8 is rounded to the nearest integer, 7):

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

Relational operators

Relational operators enable you to compare two values. The following table summarizes the relational operators:

Operator	Denomination	Description	Syntax
=	Equality operator	Compares two values and verifies their equality	`<Expression 1> = <Expression 2>`
<	Less-than operator	Tests whether a value is strictly less than another	`<Expression 1> < <Expression 2>`
<=	Less-than or equal to operator	Test whether a value is less than or equal to another	`<Expression 1> <= <Expression 2>`
>	Greater-than operator	Tests whether a value is strictly greater than another	`<Expression 1> > <Expression 2>`
>=	Greater-than or equal to operator	Tests whether a value is greater than or equal to another	`<Expression 1> >= <Expression 2>`

Operator	Denomination	Description	Syntax
<>	Inequality operator	Tests whether a value is different from another	`<Expression 1> <> <Expression 2>`

Logical operators

Logical operators enable you to evaluate several conditions.

The And operator

This operator performs a logical AND (both conditions must be true) on two expressions. The syntax is as follows:

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

If each expression (operand) is evaluated as *True*, the result is *True*. If either of the expressions is evaluated as *False*, the result is *False*.

The Or operator

This operator performs a logical OR (either of the conditions must be true) on two expressions. The syntax is as follows:

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

If either expression is evaluated as *True*, the result is *True*.

The Xor operator

This operator performs an eXclusive OR (only one of the two conditions must be true) on two expressions. The syntax is as follows:

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

If only one of the expressions is evaluated as *True*, result is *True*.

The Not operator

This operator is used to perform the logical negation on an expression. The syntax is as follows:

```
<Result> = Not <Expression 1>
```

If the expression is evaluated as *True*, the result is *False*. If the expression is evaluated as *False*, the result is *True*.

Priority of operators

When more than one operators are combined, the following order of priority is used when evaluating expressions. The operators are listed in decreasing order of priority:

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

File management

Connect-It Basic enables simplified file management. The most common operations (read, write, etc.) are available as standard.

Reminder concerning files

A file is way in which a program sees and external object. It is a collection of logical records, that may or may not be structured, on which the program can execute a set of elementary operations (read, write, etc.). A logical record represents the minimum set of data that can be manipulated by a single elementary operation.

Connect-It can only handle so-called sequential files. In a sequential file, operations mainly concern reading the next record or appending a new record to the end of the file. It is not possible to simultaneously read and write records.

When read, cursor is placed on the first logical record of the sequential file. Each read operation transfers a record to an internal zone (generally a variable) of the program and places the cursor on the next record of the file. An operation enables you to determine whether there are any remaining records to be read (**EOF** clause: End Of File).

When written to, the sequential file is either empty or the cursor is placed after the last record in the file. Each write operation transfers data stored in an internal zone (generally a variable) of the program, to a record in the file and then moves the cursor after this record.

 Note:

One of the main features of a sequential file is that the records are read in the order they are written.

Opening and closing files

The Open clause

This is the main clause used to manipulate files. It enables you to read, create and write to a files. The syntax is as follows:

```
Open <Path of the file> For <Mode> [Access <Access type>] As [#]<File numb
er>
```

The parameters of this clause are detailed in the following table:

Parameter	Description
<Path of the file>	Character string specifying the file concerned by the operation. This string can contain the full path of the file.
<Mode>	Specifies the processing mode of the file. This parameter may contain one of the following values: ■ *Input*: The file is open in read mode. ■ *Output*: The file is open in write mode. If the file already exists and has existing content, this is overwritten. ■ *Append*: The file is open in write mode. If the file already exists and has existing content, the new content is appended to the end of the file.
<Access type>	Specifies the operations than can be performed on an open file. If the file is opened by another process and the specified access is not authorized, the file open command fails. This parameter can be set to any of the following values: ■ *Read*: The file is open for read-only access ■ *Write*: The file is open for write-only access ■ *Read Write*: The file is opened in read-write mode. This type of access is only available for *Append* mode.

Parameter	Description
<File number>	Identifies the file using a unique number between 1 and 511. The **FreeFile()** function enables you to determine the next available file number.

Note:

Bear the following points in mind:

- Files must be opened using the **Open** clause before any read or write operations on the file.
- In *Append*, *Binary* or *Output* mode, if the referenced file does not exist, it is created.
- In *Binary* or *Input* mode, you can open a file using a different number without having to close the file first. In *Append* or *Output* mode, you must first close a file before opening it again with a different number.

The Close clause

This clause enables you to close a file that was opened using a the **Open()**. The syntax is as follows:

```
Close [<List of files>]
```

The optional **<List of files>** argument can contain one or more file numbers. This syntax of this optional argument is as follows:

```
[[#]<File number>][,[#]<File number>]...
```

Note:

If you omit this parameter from the clause, all active files opened by the **Open()** clause are closed.

Reading data from file

Two clauses are available for reading data from a file. Using one or the other clause will depend on the specified access mode for the file. The two clauses are the following:

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

In Input clause

This clause is used to read a given number of characters from a file open in *Binary* or *Input* mode. The syntax of this clause is as follows:

```
Input (<Number characters to read>,[#]<File number>)
```

The Line Input clause

This clause is used to read a line of data from a sequential file, and to store it in a *String* or *Variant* type variable. The syntax of this file is as follows:

```
Line Input #<File number>, <Name of the variable>
```

Important:

The clause reads the characters one by one until a carriage return or carriage return - new line is reached.

Writing data to a file

One single clause, **Print**, enables you to write data to a file. The syntax of this clause is as follows:

```
Print #<File number>, [<Data>]
```

Note:

The **Print** clause writes data to the file on a single line until a linefeed character (ASCII code 10) is encountered. For example:

```
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 Field of application

The functions described in this document can be used in all script windows in Connect-It.

3 Conventions

This chapter contains information on the notation used in this manual and on the format of some constants.

Notation

The following notation is used in the examples in this manual:

[]	Square brackets denote an optional parameter. Do not type these brackets in your command. Exception: In Basic scripts, when the square brackets denote the path to data in the database with the form: `[Link.Link.Field]`
<>	Angle brackets denote a parameter in plain language. Do not type these brackets. Substitute the text with the appropriate information.
{}	Curly brackets surround the definition of a node or a multi-lined script block for a property.
|	A pipe is used to separate a series of possible parameters contained within curly brackets.

The following text styles have specific meanings:

`Fixed width characters`	DOS command, function parameter or data formatting.
`Example`	Example of code or command.
...	Code or command omitted.
Object name	The names of fields, tabs, menus and files are shown in bold.
Note: Note	Important note.

Format of "Date+Time" constants in scripts

Dates referenced in scripts are expressed in international format, independently of the display options specified by the user:

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

Example:

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

Note:

The hyphen ("-") can also be used as a date separator.

Basic and Unix date

Dates are expressed differently in Basic and in Unix:

- In Basic, a date can be expressed in international format, or as a floating point number ("Double" type). In this case, the integer part of the number represents the number of days elapsed since 1899-12-30 at midnight, the decimal part represents the fraction of the current date. (The number of seconds elapsed since the start of the day divided by 86400.)
- In Unix, dates are expressed as a long integer ("Long" type) that represents the number of seconds elapsed since 01/01/1870 at midnight, independent of time zones (UTC time).

Format of "Duration" type constants in scripts

In scripts, durations are stored and expressed in seconds. For example, to set the default value for a "Duration" type field to 3 days, use the following script:

```
RetVal=259200
```

Likewise, functions that calculate durations return a number in seconds.

 Note:

In financial calculations, Connect-It takes into account the most common simplifications used. In this case alone, a year is considered as being 12 months and 1 month as 30 days (thus: 1 year = 360 days).

4 Definitions

This chapter groups together the definitions of several essential terms.

Definition of a function

A function is a program that performs operations and returns a value to the user. This value is called the "return value" or "return code".

Here is an example of the syntax used to call a function using the Connect-It internal Basic:

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

Definition of an error code

When a function fails, it returns an error code.

The description of the last error and its code can be found using the **Err.Description** and **Err.Number**.

You can trigger an error message intentionally using the **Err.Raise** function. Its syntax is as follows:

```
Err.Raise (<Error number>, <Error message>)
```

The following table lists the most frequent error codes:

Error code	Meaning
12001	Undefined error
12002	Bad parameter for a function
12003	Invalid handle or object deleted
12004	No more data available. This error typically occurs when executing queries. When the query does not return data, this error is raised.
12006	Invalid value (incorrect type for a parameter, etc.)
12009	Obsolete or non implemented function

5 Function typing and parameters

List of types

The following table summarizes the various types available for a function or a parameter:

Type	Description
Integer	Integer from -32,768 to +32,767.
Long	Integer from -2,147,483,647 to +2,147,483,646.
Single	4-bype floating-point number (single precision).
Double	8-byte floating-point number (double precision).
String	Text in which all characters are allowed.
Date	Date or Date+Time.
Variant	Generic type that can represent any type.

Type of a function

The type of a function corresponds to the type of the value returned by the function. We recommend paying close attention to this piece of information since it can be at the origin of compilation and runtime errors in your programs.

Type of a parameter

The parameters that can be used in functions also have a type that must be respected in order for the proper execution of the function. In the syntax of functions, parameters are prefixed according to their type. The following table resumes the various types and their associated prefixes:

Type	Prefixe used in the Basic syntax
Integer	"i"
Long	"l"
Double	"d"
String	"str"
Date	"dt"
Variant	"v"

6 Examples of scripts

This section presents sample scripts sorted according to the different elements that they use.

Basic functions

If, Then, Else, Else If, End If

Syntax

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

Note:

About logical fields (Boolean):

Logical fields are represented as 8-bit integers. The value "true" in Basic is equal to -1.

Certain scripts concerning logical fields can pose problems:

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

If the value "true" defined for your database is 1 and the value "false" is 0, then, for this script, the value returned will be 1, and "false" as understood in Basic.

Example

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

This script returns the value:

- *Empty* if the text field of a produced document doesn't contain any information.
- *Default* if the text field of a produced document contains the *default* information.
- *Unknown* if the text field of a produced document contains any other information.

For Loop

This function enables you to create a loop.

Syntax

For <counter variable> = <start> **to** <end>

<Instructions>

Next

Example

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

This script returns the value *i* in the Document log.

You will see the following in the Document log:

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

While Loop

This instruction enables you to create a loop.

Syntax

While loop

While <conditions>

<instructions>

WEnd

Example

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

This script returns the value *i* in the Document log if this value is less than 10.

You will see the following in the Document log:

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

Return

In the script, if the conditions defined before this function are not respected then the rest of the script is ignored.

Syntax

<conditions>

Return

<conditions>

Example

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

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

This script tests whether the **MacAdress** and **IPAdress** fields of a produced document have not got an empty value. If this condition is fulfilled:

- the current node is ignored
- the end of the script is not executed

Select

This function enables you to execute a block of instructions according to the value of a variable.

Syntax

Select Case <variable to test>

Case <variable 1>

Instruction block

Case <variable 2>

Instruction block

Case <variable 3>

Instruction block

...

Case <variable n>

Instruction block

Case Else

End Select

Example

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

In this example:

- The source document's **seStatus** field corresponds to the status of a ticket.
- The status of the ticket is:
  - 0 = open ticket
  - 1 = closed ticket

This script associates the character string describing the status of the ticket to the numeric value of the source field. If the status is unknown, the *Unknown Status* value is returned.

Pif functions

The **PIF** functions have been specially developed for Connect-It mapping scripts.

PifIgnoreDocumentMapping

This function enables you to ignore the processing of a document.

Syntax

<conditions>

PifIgnoreDocumentMapping("<message>")

<conditions>

("message) enables you to display an error message in the document log for the ignored element.

The specification of a *retval* function implies that the *PifIgnore* function is executed on a field chosen as a reconciliation key.

Example

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

We use the **MacAddress** field for a reconciliation key. If this field does not contain a value, the document is ignored. The message *Missing MacAddress* field is shown in the document log.

PifRejectDocumentMapping

This function enables you to reject a source document and to not send it to the destination connector.

This applies to any element of the document:

- root node
- structure
- collection
- field

Syntax

<instructions>

PifRejectDocumentMapping("message")

<instructions>

("message) enables you to display an error message in the document log for the ignored element.

The specification of a *retval* function implies that the *PifReject* function is executed on a field chosen as a reconciliation key.

Example

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

We use the **MacAddress** field for a reconciliation key. If this field does not contain a value, the document is ignored. The message *Missing MacAddress* field is shown in the document log.

PifIgnoreNodeMapping

This function enables you to ignore any element in a document type.

This element can be:

- The root node of the document
- A structure
- A collection
- A field

The behavior of the *PifIgnoreNodeMapping* function is different depending on whether it concerns a collection or not.

If this instruction concerns a collection, only the current member of the collection is ignored. If you want to ignore all members of the collection, use the *PifIgnoreCollectionMapping* instruction.

Syntax

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

("message) enables you to display an error message in the document log for the ignored element.

Example

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

This script enables you to avoid updating with an empty string if the field or the structure containing the **MAC address** field is empty. If the field is populated then the update is performed.

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

This script enables you to ignore the member of a collection if the **Sofware.Name** field of this member is set to *Windows* or *SunOS*.

PifIgnoreCollectionMapping

This function enables you to ignore a collection of a produced document-type during a collection to collection mapping.

For more information about the collection-to-collection mapping, please read the Connect-It - User's Guide, chapter *Implementing an integration scenario*.

Syntax

<instructions>

PifIgnoreCollectionMapping

<instructions>

Example

```
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
```

For a processing report, this script enables all the members of the **logs** collection to be ignored if there is no error message.

If the document does not contain an error, it is not necessary to carry out such a script. The **ErrorNumber** field contains the number of errors associated to a document.

The previous script can be replaced by the following:

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

Collections

In this section, you will find different examples of scripts concerning the processing of collections.

Creating members in a collection from a list of values

This section presents a script example enabling you to create a member in a given collection from a list of values from a source document.

In this example:

- This **Software** source field contains a list of values.
- The values are separated by a given separator.

The script:

- Extracts the software names one by one.
- Creates a member in the **SoftInstalled** destination collection.
- Populates the **Name** element with the name of the extracted software.

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

```
Dim strPath As String

' Count of number of values in the "Software" source field
' the software names are separated by the separator (','), for example: "E
xcel, Connect-It,
' Asset Manager"
iCount = CountValues([Software], ",")

' Loops around all the elements in the list to extract them one by one.
For iIndex = 0 To iCount - 1

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

' Deletion of spaces around the name of the software
strSoft = Trim(strSoft)

' Creation of the path of the destination collection from the root element
.
' For example, for the third source software, the path "SoftInstalled(3).N
ame" is created
strPath = "SoftInstalled" (& iIndex & ").Name"

' Assigning of the current value of character string software to the path
using
' the function PifSetStringVal.
' The function PifSetStringVal returns an error code if the path is not va
lid, it is
' necessary to assign in a variable the return value of the function. The
function will not
' be applied in the opposite case.
lDummy = PifSetStringVal(strPath, strSoft)

Next iIndex
```

This mapping script can be applied on any destination-document type element.

To better read the mapping, we recommend that you do the mapping on the collection to which the members must be added.

Warning:

The element indicated by its path when calling on the Basic function **PifSetStringVal** must be present in the destination document-type. In the present example, the **Name** element of the **SoftInstalled** collection must be added by the user in the consumed document-type.

Concatenating members of a collection in a field

In this example:

- The source document contains a collection of values.
- This element's collections are mapped to a destination document-type field.

The source contains the collection of software installed on a computer. The different names of the software must be written in a field containing the list of software, separated by a comma (',').

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

Dim strList As String
Dim iItem As Integer

For each element in the collection, recover the name of the software (Elem
ent "Name" of the collection "SoftInstalled") and concatenate it with the
current list.
For iItem = 0 to iCollectionCount - 1

' Add the name separator if the list is not empty
If strList = "" Then
strList = strList & ", "

' Add the name of the software to the current list.
' Note that it is possible to directly use a variable to indicate the numb
er
' of a member in a collection. For example, if the variable of iITem is 3,
the path
' [SoftInstalled(3).Name] will automatically be created from the value of
iItem.
  strList = strList & ", " [SoftInstalled(iItem).Name]
Next iItem

' Assign the variable strList to the target element
RetVal = strList
```

Mapping several fields in a collection

In this example:

- The source document contains several distinct fields.

  Here, the *Address1* and *Address2* have the two possible addresses of a client.
- The value of these fields must be associated to a member of the destination collection.

  Here, the collection *Address*.

For example:

You must therefore:

- Create two members in the destination collection and associated them to the "Adress1" and "Address2" fields.
- Use the collection-duplication function:
  1. Add the *Address* collection to the destination document-type.
  2. Duplicate this collection.

     The *Address#1* collection appears in the destination document-type.

3 The mapping scripts [Address1] and [Address2] must be applied to the fields *Address.Address* and *Address#1.Address*, respectively.

Ignoring certain members of a collection in a collection-to-collection mapping

To ignore certain members in a collection, you must use the *PifIgnoreCollectionMapping* and *PifIgnoreNodeMapping* instructions.

For more information about these instructions, refer to the section PifIgnoreCollectionMapping [page 45].

Script concerning a connector not included in a mapping

The following example describes the integration in a scenario (which concerns the replication of data between a database and a ServiceCenter database) of an Asset Manager database. The script imports an employee. During the import, the script verifies whether the employee exists in the Asset Manager and changes the mapping accordingly.

1 Add an Asset Manager connector to your scenario. This connector is not required to be linked to a mapping box or another connector, just its title is important (**Connector name** field of the connector configuration wizard) because it will be used in the script. Here, the connector is called Asset Management.

2 Create a new document type produced by the Asset Manager connector. Select the Departments and Employees table (**amEmplDept**) and call the produced document type (**Document type** field) **amEmplDeptForMapping**. This name will be used in the script.

> Note:
>
> If you define WHERE or ORDER BY clauses, they are not taken into account in the sample script.

3 In the mapping box, populate the script field as follows:

```
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)
```

Syntax of the pifNewQueryFromFmtName function

This function creates a query on a document type first defined in the list of documents produced by a resource.

The parameters of the function are as follows:

- **strCntrName**: This parameter contains the name of the resource (on which the query is performed).
- **strFmtName**: This parameter contains the name of the document type (that has been defined as the produced document type).
- **strLayer**: This parameter contains the production directives (for example, a WHERE clause).

The function returns a handle (here, the **hQuery** parameter). This handle must be passed as a parameter to the **PifQueryNext** in order to browse the list of returned records.

The data from the current document can then be recovered using one of the following functions (depending on the field type):

- pifQueryGetStringVal
- pifQueryGetDateVal
- pifQueryGetDoubleVal
- pifQueryGetLongVal
- pifQueryGetIntVal

Each of these functions has two parameters:

- The handle (hQuery) of the query to use (32-bit long integer)
- Path of the element for which we want to recover a value. This path must not contain the name of the root element of the document type (**amEmplDept** in this example).

  In this example, the function returns the name of the employee:

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

Production directives of the pifNewQueryFromFmtName function

The **pifNewQueryFromFmtName** function uses simple parameters. You can, however, define complex queries in XML format.

The production directives can be given in XML, using the following syntax:

```
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)
```

XML syntax

The &, < and > characters are not authorized. You must replace them with &, < and > respectively. The Basic function **GetXmlElementValue** handles the substitution of these characters.

For example:

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

Query on fields containing a period or comma

In the following commented example, a query involves the elements **mac.address** and **logical.name** of an HP Network Discovery - ServiceCenter scenario. The script validates the MAC address provided by the IND connector before assigning it a reconciliation key. If the MAC address is validated, the information from the **logical.name** field is recovered instead of from the **mac.address** field.

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

strQuery = "mac.address = " & chr(34) & [MACAddress] & chr(34)
"MAC address in the PDI document

hQuery = pifNewQueryFromFmtName("ServiceCenter", "pc1", strQuery)
"pc1 is the document produced for the ServiceCenter Computers table

Dim strValue as String
strValue = [MACAddress]
"strValue by defaut

iRc = pifQueryNext(hQuery)
if iRc = 0 then         "
"query finished because iRc=0

strValue = pifQueryGetStringVal(hQuery, "'logical.name'")
"Single quotes define the parameter logical.name as a field and not a path
```

```
pifLogWarningMsg("Matched Asset using query: " & strQuery)
"write to document log

pifLogWarningMsg("Updating Asset " & strValue)
"strValue is not written to the document log
else
pifLogWarningMsg("Could not locate existing asset using MAC address " & [M
acAddress])
end if

iRc = pifQueryClose(hQuery)

If strValue = "" then
"This code is executed when pifQueryNext returns 0.

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

II Reference

7 Alphabetic reference

Abs()

Internal Basic syntax

Function Abs(dValue As Double) As Double

Description

Returns the absolute value of a number.

Input parameters

- **dValue**: Number for which you want to know the absolute value.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

AppendOperand()

Internal Basic syntax

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

Description

Concatenates a string according to the parameters passed to the function. The results are given as follows:

```
strExpr strOperator strOperand
```

Input parameters

- **strExpr**: Expression to be concatenated.
- **strOperator**: Operator to concatenate.
- **strOperand**: Operand to concatenate.

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

If one of the **strExpr** or **strOperand** parameters is omitted, **strOperator** is not used in the concatenation.

ApplyNewVals()

Internal Basic syntax

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

Description

Assigns identical values to identical cells in a "ListBox" control.

Input parameters

- **strValues**: Source string containing the values of a "ListBox" control to be processed.
- **strNewVals**: New value to assign to the cells concerned.
- **strRows**: Identifiers of lines to be processed. The identifiers are separated by commas.
- **strRowFormat**: Formatting instructions for the sublist. Instructions are separated by the "|" character. Each instruction represents the number of the column containing the **strNewVals** parameter.

Output parameters

In case of error, a message is written to the Connect-It log.

Asc()

Internal Basic syntax

Function Asc(strAsc As String) As Long

Description

Returns a numeric value that is the ASCII code for the first character in a string.

Input parameters

- **strAsc**: Character sting on which the function operates.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Atn()

Internal Basic syntax

Function Atn(dValue As Double) As Double

Description

Returns the arc tangent of a number, expressed in radians.

Input parameters

- **dValue**: Number for which you want to know the arc tangent.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

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

BasicToLocalDate()

Internal Basic syntax

Function BasicToLocalDate(strDateBasic As String) As String

Description

This function converts a Basic format date to a string format date (as displayed in Windows Control Panel).

Input parameters

- **strDateBasic**: Date in Basic format to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

BasicToLocalTime()

Internal Basic syntax

Function BasicToLocalTime(strTimeBasic As String) As String

Description

This function converts a Basic format time to a string format time (as displayed in Windows Control Panel).

Input parameters

- **strTimeBasic**: Time in Basic format to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

BasicToLocalTimeStamp()

Internal Basic syntax

Function BasicToLocalTimeStamp(strTSBasic As String) As String

Description

This function converts a Date+Time in Basic format to a Date+Time in string format (as displayed in Windows Control Panel).

Input parameters

- **strTSBasic**: Date+Time in Basic format to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Beep()

Internal Basic syntax

Function Beep()

Description

Plays a beep on the machine.

Output parameters

In case of error, a message is written to the Connect-It log.

CDbl()

Internal Basic syntax

Function CDbl(dValue As Double) As Double

Description

Converts an expression to a "Double".

Input parameters

- **dValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

ChDir()

Internal Basic syntax

Function ChDir(strDirectory As String)

Description

Changes the current directory.

Input parameters

- **strDirectory**: New current directory.

Output parameters

In case of error, a message is written to the Connect-It log.

ChDrive()

Internal Basic syntax

Function ChDrive(strDrive As String)

Description

Changes the current drive.

Input parameters

- **strDrive**: New drivename.

Output parameters

In case of error, a message is written to the Connect-It log.

Chr()

Internal Basic syntax

Function Chr(lChr As Long) As String

Description

Returns a string corresponding to the ASCII passed by the **iChr** parameter.

Input parameters

- **lChr**: ASCII code of the character.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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()

Internal Basic syntax

Function CInt(iValue As Long) As Long

Description

Converts any valid expression to an Integer.

Input parameters

- **iValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

CLng()

Internal Basic syntax

Function CLng(lValue As Long) As Long

Description

Converts any valid expression to a Long.

Input parameters

- **lValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Cos()

Internal Basic syntax

Function Cos(dValue As Double) As Double

Description

Returns the cosine of a number, expressed in radians.

Input parameters

- **dValue**: Number whose cosine you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Notes

Note:

The conversion formula for degrees to radians is the following:

```
angle in radians = (angle en degrees) * Pi / 180
```

CountOccurences()

Internal Basic syntax

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

Description

Counts the number of occurrences of a string inside another string.

Input parameters

- **strSearched**: Character string in which to perform to the search.
- **strPattern**: Character string to find inside the **strSearched** parameter.
- **strEscChar**: Escape character. If the function encounters this character inside the **strSearched** string, the search stops.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=CountOccurences("you|me|you,me|you", "you", ",") :'Returns "2"
MyStr=CountOccurences("you|me|you,me|you", "you", "|") :'Returns "1"
```

CountValues()

Internal Basic syntax

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

Description

Counts the number of elements in a string, taking into account a separator and an escape character.

Input parameters

- **strSearched**: Character string to process.
- **strSeparator**: Separator used to delimit the elements.
- **strEscChar**: Escape character. If this character prefixes a separator, it will be ignored.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=CountValues("you|me|you\|me|you", "|", "\") :'Returns 4
MyStr=CountValues("you|me|you\|me|you", "|", "") :'Returns 5
```

CSng()

Internal Basic syntax

Function CSng(fValue As Single) As Single

Description

Converts any valid expression to a floating point number ("Float").

Input parameters

- **fValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

CStr()

Internal Basic syntax

Function CStr(strValue As String) As String

Description

Converts any valid expression to a String.

Input parameters

- **strValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

CurDir()

Internal Basic syntax

Function CurDir() As String

Description

Returns the current path.

Output parameters

In case of error, a message is written to the Connect-It log.

CVar()

Internal Basic syntax

Function CVar(vValue As Variant) As Variant

Description

Converts any valid expression to a Variant.

Input parameters

- **vValue**: Expression to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Date()

Internal Basic syntax

Function Date() As Date

Description

Returns the current system date.

Output parameters

In case of error, a message is written to the Connect-It log.

DateAdd()

Internal Basic syntax

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

Description

This function calculates a new date according to a start date to which a real duration is added.

Input parameters

- **tmStart**: This parameter contains the date to which the duration is added.
- **tsDuration**: This parameter contains the duration (expressed in seconds) to be added to the date **tmStart**.

Output parameters

In case of error, a message is written to the Connect-It log.

DateAddLogical()

Internal Basic syntax

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

Description

This function calculates a new date according to a start date to which a logical duration is added (1 month contains 30 days).

Input parameters

- **tmStart**: This parameter contains the date to which the duration is added.
- **tsDuration**: This parameter contains the duration, expressed in seconds, to be added to the date **tmStart**.

Output parameters

In case of error, a message is written to the Connect-It log.

DateDiff()

Internal Basic syntax

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

Description

This function calculates in the seconds the duration (or timespan) between two dates.

Input parameters

- **tmEnd**: This parameter contains the end date of the period for which the calculation is carried out.
- **tmStart**: This parameter contains the start date of the period for which the calculation is carried out.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example calculates the time elapsed between 01/01/98 and 01/01/99.

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

DateSerial()

Internal Basic syntax

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

Description

This function returns a date formatted according to the **iYear**, **iMonth** and **iDay** parameters.

Input parameters

- **iYear**: Year. If the value is between 0 and 99, this parameter describes the years from 1900 to 1999. For all other years, you must specify the four digits (e.g. 1800).
- **iMonth**: Month.
- **iDay**: Day.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

Each of these parameters can be set to a numeric expression representing a number of days, months or years. For example:

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

Returns the value:

```
1989/1/7
```

When the value of a parameter is out of the expected range (i.e. 1-31 for days, 1-12 for months, etc.), the function returns an empty date.

DateValue()

Internal Basic syntax

Function DateValue(tmDate As Date) As Date

Description

This function returns the date portions of a "Date+Time" value.

Input parameters

- **tmDate**: "Date+Time" format date.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

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

Returns the value:

```
1999/09/24
```

Day()

Internal Basic syntax

Function Day(tmDate As Date) As Long

Description

Returns the day contained in the **tmDate** parameter.

Input parameters

- **tmDate**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

EscapeSeparators()

Internal Basic syntax

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

Description

Prefixes one or more separator characters with an escape character.

Input parameters

- **strSource**: Character string to process.
- **strSeparators**: List of separators to be prefixed. If you want to declare several separators, you must separate them with the escape character (indicated in the **strEscChar** parameter.
- **strEscChar**: Escape character. It will be used to prefix all separators in **strSeparators**.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=EscapeSeparators("you|me|you,me|you", "|\,", "\") :'Returns "you\|me
\|you\,me\|you"
```

ExeDir()

Internal Basic syntax

Function ExeDir() As String

Description

This function returns the full path of the executable.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Exp()

Internal Basic syntax

Function Exp(dValue As Double) As Double

Description

Returns the exponent of a number.

Input parameters

- **dValue**: Number whose exponent you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

ExtractValue()

Internal Basic syntax

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

Description

Extracts from a string the values delimited by a separator. The recovered value is deleted from the source string. This operation takes into account a possible escape character. If the separator is not found in the source string, the whole string is returned and the source string is deleted in full.

Input parameters

- **pstrData**: Source string to be processed.
- **strSeparator**: Character used as separator in the source string.
- **strEscChar**: Escape character. If this character prefixes the separator, it will be ignored.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=ExtractValue("you,me", ",", "\") :'Returns "you" and leaves "me" in
the source string
MyStr=ExtractValue(",you,me", ",", "\") :'Returns "" and leaves "you,me" i
n the source string
MyStr=ExtractValue("you", ",", "\") :'Returns "you" and leaves "" in the s
ource string
MyStr=ExtractValue("you\,me", ",", "\") :'Returns "you\,me" and leaves ""
in the source string
MyStr=ExtractValue("you\,me", ",", "") :'Returns "you\" and leaves "me" in
the source string
RetVal=""
```

FileCopy()

Internal Basic syntax

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

Description

Copies a file or a folder.

Input parameters

- **strSource**: Full path of the file or directory to copy.
- **strDest**: Full path of the target file or directory.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

FileDateTime()

Internal Basic syntax

Function FileDateTime(strFileName As String) As Date

Description

Returns the time and date of a file as a Long.

Input parameters

- **strFileName**: Full path name of the file concerned by the operation.

Output parameters

In case of error, a message is written to the Connect-It log.

FileExists()

Internal Basic syntax

Function FileExists(strFileName As String) As Long

Description

This function tests for the existence of a file. The function returns the following values:

- 0: File not found.
- 1: File found.

Input parameters

- **strFileName**: This parameter contains the full path of the file you want to test for.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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()

Internal Basic syntax

Function FileLen(strFileName As String) As Long

Description

Returns the size of a file.

Input parameters

- **strFileName**: Full path name of the file concerned by the operation.

Output parameters

In case of error, a message is written to the Connect-It log.

Fix()

Internal Basic syntax

Function Fix(dValue As Double) As Long

Description

Returns the integer portion of a number (first greatest integer in the case of a negative number).

Input parameters

- **dValue**: Number whose integer portion you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

FormatDate()

Internal Basic syntax

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

Description

Formats a date according to the expression contained in the **strFormat** parameter.

Input parameters

- **tmFormat**: Date to be formatted.
- **strFormat**: Expression containing the formatting instructions.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example of code shows how to format a date:

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

FormatResString()

Internal Basic syntax

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

Description

This function processes a source string, replacing the variable $1, $2, $3, $4, and $5 with the strings passed in the **strParamOne**, **strParamTwo**, **strParamThree**, **strParamFour**, and **strParamFive** parameters.

Input parameters

- **strResString**: Source string to be processed.
- **strParamOne**: Replacement string of variable $1.
- **strParamTwo**: Replacement string of variable $2.
- **strParamThree**: Replacement string of variable $3.
- **strParamFour**: Replacement string of variable $4.
- **strParamFive**: Replacement string of variable $5.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

```
FormatResString("I$1he$2you$3", "you", "we", "they")
```

returns "Iyouheweyouthey".

FV()

Internal Basic syntax

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

Description

This function returns the future amount of an annuity based on constant and periodic payments, with a set interest rate.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

- **iNper**: This parameter contains the total number of dates of payment for the financial operation.
- **dblPmt**: This parameter indicates the amount of the payment to be made at each date of payment. The payment generally includes both principal and interest.
- **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

- The **Rate** and **Nper** parameters must be calculated using payments expressed in the same units.
- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

GetEnvVar()

Internal Basic syntax

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

Description

This function returns the value of an environment variable. An empty value is returned if the environment variable does not exist.

Input parameters

- **strVar**: This parameter contains the name of the environment variable.
- **bExpand**: This Boolean parameter is useful when the environment variable references one or more environment variable. In this case, when this parameter is set to 1 (default value), each referenced variable is replaced by its value. Otherwise, it is left alone.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

GetListItem()

Internal Basic syntax

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

Description

Returns the **lNb**th portion of a string delimited by separators.

Input parameters

- **strFrom**: Source string to be processed.
- **strSep**: Character used as separator in the source string.
- **lNb**: Position of the string to recover.
- **strEscChar**: Escape character. If this character prefixes a separator, it will be ignored.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

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

returns "is".

```
GetListItem("this%_is_a_test", "_", 2, "%")
```

returns "a".

GetXmlAttributeValue()

Internal Basic syntax

Function GetXmlAttributeValue(strVal As String) As String

Description

This function escapes the invalid characters for a given XML element value.

Input parameters

This parameter contains the character to be escaped.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

For example, the following function transforms the single quote character into a valid character:

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

It has the following result:

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

GetXmlElementValue()

Internal Basic syntax

Function GetXmlElementValue(strElemContent As String) As String

Description

This function escapes the invalid characters for a given XLM attribute value.

Input parameters

This parameter contains the character to be escaped.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

You can use this function in a WHERE clause as described below:

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

See also the example with comments for the function:
PifNewQueryFromFmtName() [page 150]

Hex()

Internal Basic syntax

Function Hex(dValue As Double) As String

Description

Returns the hexadecimal value of a decimal parameter.

Input parameters

- **dValue**: Decimal number whose hexadecimal value you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Hour()

Internal Basic syntax

Function Hour(tmTime As Date) As Long

Description

Returns the hour value contained in the **tmTime** parameter.

Input parameters

- **tmTime**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

InStr()

Internal Basic syntax

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

Description

Returns the character position of the first occurrence of a string within a string.

Input parameters

- **lPosition**: Starting point of the search. This parameter is not optional and must be a valid positive integer no greater than 65,535.
- **strSource**: String in which the search is performed.
- **strPattern**: String to search.
- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0).

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Notes

Note:

The position of the first occurence is always 1. The function returns 0 if the character string searched is not found.

Int()

Internal Basic syntax

Function Int(dValue As Double) As Long

Description

Returns the integer portion of a number (first lesser than integer in the case of a negative number).

Input parameters

- **dValue**: Number whose integer portion you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

IPMT()

Internal Basic syntax

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

Description

This function returns the amount of interest for an given date of payment of an annuity.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

  - **iPer**: This parameter indicates the period for the calculation, between 1 and the value of the **Nper** parameter.
  - **iNper**: This parameter contains the total number of dates of payment for the financial operation.
  - **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.

- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

- The **Rate** and **Nper** parameters must be calculated using payments expressed in the same units.
- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

IsNumeric()

Internal Basic syntax

Function IsNumeric(strString As String) As Long

Description

This function enables you to determine whether a string is a numeric value or not.

Input parameters

- **strString**: This parameter contains the character string to evaluate.

Output parameters

In case of error, a message is written to the Connect-It log.

Kill()

Internal Basic syntax

Function Kill(strKilledFile As String) As Long

Description

Deletes a file.

Input parameters

- **strKilledFile**: Full path of the file concerned by the operation.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

LCase()

Internal Basic syntax

Function LCase(strString As String) As String

Description

Returns a string in which all letters of the string parameter have been converted to lower case.

Input parameters

- **strString**: Character string to convert to lowercase.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
' 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()

Internal Basic syntax

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

Description

Returns the left most iNumber characters of a string parameter.

Input parameters

- **strString**: Character string to process.
- **lNumber**: Number of characters to return.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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()

Internal Basic syntax

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

Description

Extracts the portion of a string to the left of the separator specified in the **strSep** parameter.

The search for the separator is performed from left to right.

The search can be made case sensitive using the **bCaseSensitive** parameter.

Input parameters

- **strFrom**: Source string to be processed.
- **strSep**: Character used as separator in the source string.
- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0).

Output parameters

In case of error, a message is written to the Connect-It log.

Example

These examples illustrate use of the **LeftPart**, **LeftPartFromRight**, **RightPart**, and **RightPartFromLeft** functions on the same string: "This_is_a_test":

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

Returns "This".

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

Returns "This_is_a".

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

Returns "test".

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

Returns "is_a_test".

LeftPartFromRight()

Internal Basic syntax

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

Description

Extracts the portion of a string to the left of the separator specified in the **strSep** parameter.

The search for the separator is performed from right to left.

The search can be made case sensitive using the **bCaseSensitive** parameter.

Input parameters

- **strFrom**: Source string to be processed.
- **strSep**: Character used as separator in the source string.
- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0).

Output parameters

In case of error, a message is written to the Connect-It log.

Example

These examples illustrate use of the **LeftPart**, **LeftPartFromRight**, **RightPart**, and **RightPartFromLeft** functions on the same string: "This_is_a_test":

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

Returns "This".

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

Returns "This_is_a".

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

Returns "test".

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

Returns "is_a_test".

Len()

Internal Basic syntax

Function Len(vValue As Variant) As Long

Description

Returns the number of characters in a string or a variant.

Input parameters

- **vValue**: Variant concerned by the operation.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strTest as String
Dim iLength as Integer
strTest = "Peregrine Systems"
```

```
iLength = Len(strTest) :'The value of iLength is 17
RetVal=iLength
```

LocalToBasicDate()

Internal Basic syntax

Function LocalToBasicDate(strDateLocal As String) As String

Description

This function converts a string format date (as displayed in Windows Control Panel) to a Basic format date .

Input parameters

- **strDateLocal**: Date as string to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

LocalToBasicTime()

Internal Basic syntax

Function LocalToBasicTime(strTimeLocal As String) As String

Description

This function converts a string format time (as displayed in Windows Control Panel) to a Basic format time.

Input parameters

- **strTimeLocal**: Time in string format to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

LocalToBasicTimeStamp()

Internal Basic syntax

Function LocalToBasicTimeStamp(strTSLocal As String) As String

Description

This function converts a Date+Time in string format (as displayed in Windows Control Panel) to a Date+Time in Basic format.

Input parameters

- **strTSLocal**: Date+Time in string format to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

LocalToUTCDate()

Internal Basic syntax

Function LocalToUTCDate(tmLocal As Date) As Date

Description

This function converts a date in "Date+Time" format to a UTC format date (time-zone independent).

Input parameters

- **tmLocal**: "Date+Time" format date.

Output parameters

In case of error, a message is written to the Connect-It log.

Log()

Internal Basic syntax

Function Log(dValue As Double) As Double

Description

Returns the natural log of a number.

Input parameters

- **dValue**: Number whose logarithm you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

LTrim()

Internal Basic syntax

Function LTrim(strString As String) As String

Description

Removes all leading spaces in a string.

Input parameters

- **strString**: Character string to process.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
' 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()

Internal Basic syntax

Function MakeInvertBool(lValue As Long) As Long

Description

This function returns an inverse Boolean; (0 becomes 1, all other numbers become 0).

Input parameters

- **lValue**: Number concerned by the operation.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyValue
MyValue=MakeInvertBool(0) :'Returns 1
MyValue=MakeInvertBool(1) :'Returns 0
MyValue=MakeInvertBool(254) :'Returns 0
```

Mid()

Internal Basic syntax

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

Description

Returns a substring within a string.

Input parameters

- **strString**: String concerned by the operation.
- **lStart**: Start position of the string to extract from within strString.
- **lLen**: Length of the string to extract.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Minute()

Internal Basic syntax

Function Minute(tmTime As Date) As Long

Description

Returns the number of minutes contained in the time expressed in the **tmTime** parameter.

Input parameters

- **tmTime**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strMinute
strMinute=Minute(Date())
RetVal=strMinute :'Returns the number of minutes elapsed in the current ho
ur, for example "45" if the time is 15:45:30
```

MkDir()

Internal Basic syntax

Function MkDir(strMkDirectory As String) As Long

Description

Creates a new directory.

Input parameters

- **strMkDirectory**: Full path of the directory to create.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

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

Month()

Internal Basic syntax

Function Month(tmDate As Date) As Long

Description

Returns the month contained in the date expressed in the **tmDate** parameter.

Input parameters

- **tmDate**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim lMonth as Long
lMonth=Month(Date())
RetVal=lMonth :'Returns the current month
```

Name()

Internal Basic syntax

Function Name(strSource As String, strDest As String)

Description

Changes the name of file.

Input parameters

- **strSource**: Full path of the file to rename.
- **strDest**: New file name.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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()

Internal Basic syntax

Function Now() As Date

Description

Returns the current date and time.

Output parameters

In case of error, a message is written to the Connect-It log.

NPER()

Internal Basic syntax

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

Description

This function returns the number of payments of an annuity based on constant and periodic payments, and at a constant interest rate.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

- **dblPmt**: This parameter indicates the amount of the payment to be made at each date of payment. The payment generally includes both principal and interest.
- **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.
- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

Oct()

Internal Basic syntax

Function Oct(dValue As Double) As String

Description

Returns the octal value of the decimal parameter.

Input parameters

- **dValue**: Number whose octal value you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

ParseDate()

Internal Basic syntax

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

Description

This function converts a date expressed as a character string to a Basic date object.

Input parameters

- **strDate**: Date in string format.
- **strFormat**: This parameter contains the format of the date contained in the character string. The possible values are the following:
  - DD/MM/YY
  - DD/MM/YYYY
  - MM/DD/YY
  - MM/DD/YYYY
  - YYYY/MM/DD
  - Date: date expressed according to the settings of the client computer.
  - DateInter: date expressed in the international format
- **strStep**: This optional parameter contains the date separator used in the character string. The authorized separators are "\" and "-".

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim dDate as date
dDate=ParseDate("2001/05/01", "YYYY/MM/DD")
```

ParseDMYDate()

Internal Basic syntax

Function ParseDMYDate(strDate As String) As Date

Description

This function returns a Date object (as understood in Basic) from a date formatted as follows:

```
dd/mm/yyyy
```

Input parameters

- **strDate**: Date stored as a string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

ParseMDYDate()

Internal Basic syntax

Function ParseMDYDate(strDate As String) As Date

Description

This function returns a Date object (as understood in Basic) from a date formatted as follows:

```
mm/dd/yyyy
```

Input parameters

- **strDate**: Date stored as a string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

ParseYMDDate()

Internal Basic syntax

Function ParseYMDDate(strDate As String) As Date

Description

This function returns a Date object (as understood in Basic) in yyyy/mm/dd format.

Input parameters

- **strDate**: Date stored as a string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

PifCloseODBCDatabase()

Internal Basic syntax

Function PifCloseODBCDatabase(strDSN As String) As Long

Description

This function terminates an ODBC connection.

Input parameters

- **strDSN**: Name of the data source for the ODBC connection.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
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()

Internal Basic syntax

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

Description

This function is used to create a dynamic maptable when a session is started.

Input parameters

- **strCntrName**: Connector name.
- **strFmtName**: Document type to produce that will define the contents of the maptable.
- **strLayer**: Where clause used for the document type.
- **strMaptableName**: Name of the maptable to create.
- **strKeyName**: Column name that is used as key for the maptable to create.

  If no column name is specified as key, the first attribute of the query used to create the maptable is used.

  For example, to use the identifier of a portfolio item, the syntax will be as follows:

  ```
  Portfolio.AssetTag
  ```

  You do not need to specify the name of the document type in this syntax as the maptable is used for the current document type.
- **bCreateOnce**:
  - **0**: The maptable is created each time the function is called.
  - **1**: The maptable is created for the entire session.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

See also:

- PifMapValueEx() [page 148]
- PifMapValueContainingEx() [page 146]

PifDateToTimezone()

Internal Basic syntax

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

Description

This function converts a date (as in date and time) expressed in a given time zone to another time zone taking into account, if necessary, daylight saving settings.

An error message is returned if the specified time zone does not exist.

 Note:

The 'UTC' time zone (Universal Time Coordinated) can be used to specify a GMT 0 date without daylight savings.

Input parameters

- **tmSrc**: This parameter contains the date to be converted.
- **strTmznSrc**: This parameter contains the name of the source time zone for the conversion.
- **bWithDayLightSavingSrc**: Depending on the value of this parameter, the function will take into account (=1) or not (=0) daylight savings settings for the source time zone.
- **strTmznDst**: This parameter contains the name of the target time zone for the conversion.
- **bDayLightSavingDst**: Depending on the value of this parameter, the function will take into account (=1) or not (=0) daylight savings settings for the target time zone.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

returns:

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

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

returns:

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

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

returns:

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

Notes

List and description of time zones:

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
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

Nom	Description
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
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

Nom	Description
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()

Internal Basic syntax

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

Description

This function executes an SQL query on an ODBC database.

Input parameters

- **strDSN**: Name of the ODBC data source for the connection.
- **strSqlQuery**: SQL query to execute on the ODBC database.

Output parameters

The function returns the first result for the query. If no result matches the query, it returns a NULL value.

Example

```
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()

Internal Basic syntax

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

Description

This function returns the number of the first item in a collection according to the condition specified in **strChildCond**.

Input parameters

- **strPathCol**: Path of the collection being searched.
- **strChildCond**: Search condition on the element. The condition is in the following form:

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

  where:
  - <*Chemin*> is the path to an element in the collection
  - <*Valeur*> is the value given as a character string in the same locale as Connect-It

- Note:

  If n is the number of elements in the collection, **iStartCount** must be between 0 and n-1.

  iStartCount: Search start index.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
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()

Internal Basic syntax

Function PifGetBlobSize(strPath As String) As Long

Description

This function returns the size of a blob object.

Input parameters

- **strPath**: This parameter contains the full path of the blob object in the collection.

Output parameters

The function returns the size of a blob object identified by its full path.

Example

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

PifGetDateVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and stores it in a date type basic variable.

Input parameters

strPath: Designates a date type node type document.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into date type format.

Example

```
Dim DateVal As Date
DateVal = PifGetDateVal("field containing a date")
RetVal = DateVal
```

PifGetDoubleVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to a double.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into a double type number.

Example

```
RetVal = PifGetDoubleVal("field containing a double type number")
```

PifGetElementChildName()

Internal Basic syntax

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

Description

This function returns the name of the nth sub-element of an identified node of a source document type.

Input parameters

- **strPath**: This parameter contains the full path of the node concerned by the operation.
- **iItem**: This parameter contains the number of the sub-element whose name you want to recover.

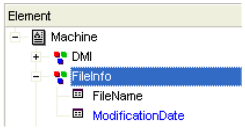
Output parameters

This function returns the name of the element.

> Note:
>
> An error is logged in the tracking lines and an empty string is returned by the function if the parameter **iItem** is either negative or greater than the number of sub-elements of the node.

Example

With the following source document type:




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

returns "FileName"

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

returns "ModificationDate"

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

returns an empty string and logs an error in the tracking lines.

PifGetElementCount()

Internal Basic syntax

Function PifGetElementCount(strPath As String) As Long

Description

This function returns the number of sub-elements of an identified node of a source document type.

Input parameters

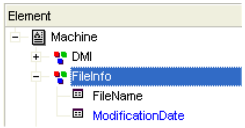
- **strPath**: This parameter contains the full path of the node concerned by the operation.

Output parameters

The function returns the number of sub-items of the node whose path is specified by **strPath**.

Example

With the following source document type:



The following script:

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

returns 2. The **FileInfo** element has 2 sub-elements: **FileName** and **ModificationDate**.

PifGetHexStringFromBlob()

Internal Basic syntax

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

Description

This function enables you to store a binary long object (blob) of the source document inside an hexadecimal string. Each bit of the binary object is stored in its hexadecimal form (two characters) inside the string. For example, the hexadecimal representation of the decimal value "27" is "1B".

Input parameters

- **strPath** : This parameter contains the full path of the binary element (blob) in the source document.
- **bCkeckNodeExists** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) if the binary element cannot be found in the source document. FALSE is the default value for this parameter.

Output parameters

The function returns the hexadecimal representation of the binary element.

Example

On some LDAP servers, an entry is uniquely defined by the binary element "ObjectGUID". The script below imports this binary value inside a string of the destination connector and uses it as a reconciliation key.

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

PifGetInstance()

Internal Basic syntax

Function PifGetInstance() As String

Description

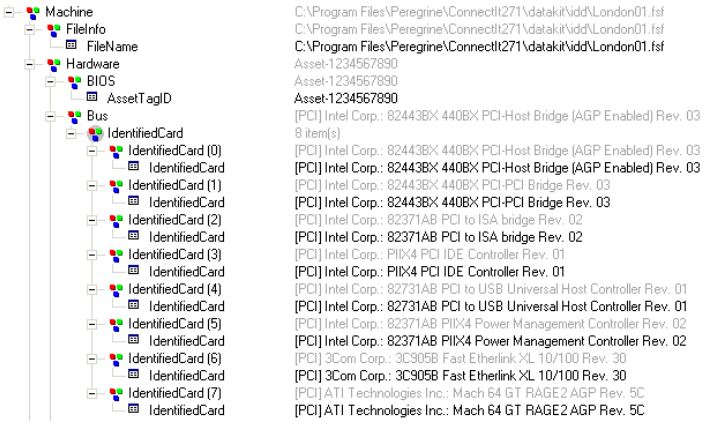
In a collection to collection mapping, this function returns the number of the element currently being processed in the collection. The first element processed is number "0".

Output parameters

The number of the element being processed is returned as a character string

Example

With the following collection mapping (on **Hardware.Bus.IdentificationCard**) :



the following script:

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

returns the following values: **Item #0**, **Item #1**, **Item #2**, etc.

PifGetIntlStringVariantVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to an international string.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value that is converted into the international string format.

Example

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

This function takes the 64-bit integer, converts it to string format and stores the result in strTest. The value displayed in the log is 281478209994880 which corresponds to 2.8147820999488e+014 processed as a double precision number by a Basic function (CStr).

Notes

This function is used to work around a known issue with the Basic engine that does not support 64-bit integers.

PifGetIntVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to an integer.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into an integer.

Example

```
Dim IValue As Int
IValue = PifGetIntVal("field containing an integer")
RetVal = IValue
```

PifGetItemCount()

Internal Basic syntax

Function PifGetItemCount(strPath As String) As Long

Description

This function returns the number of elements in a fully identified collection.

Input parameters

- **strPath**: Full path of the collection.

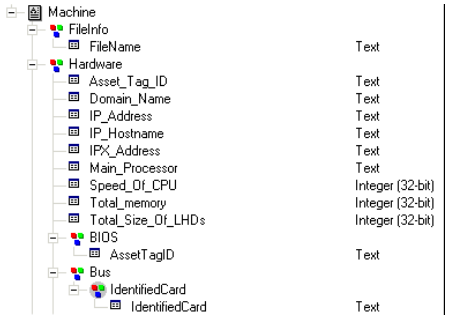
Output parameters

The function returns the number of elements in the collection.

If the collection does not exist the function will return "0".

Example

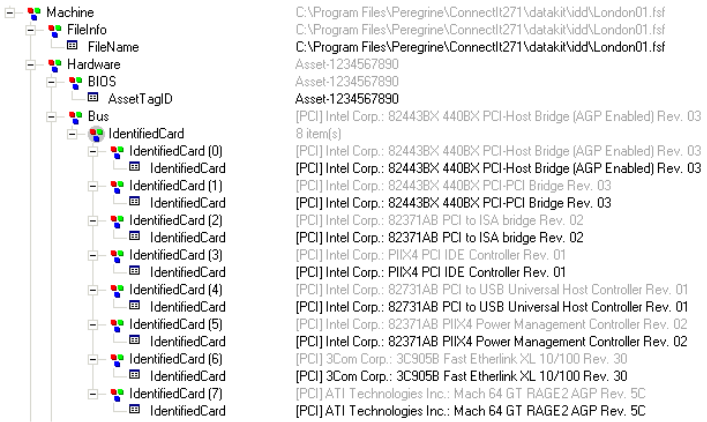
With the following document type:



The script:

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

returns the number of sub-elements of the **Hardware.Bus.IdentifiedCard** collection. For example, on the document below, the script returns "8".



PifGetLongVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to a long integer.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into an integer.

Example

```
Dim IValue As Long
IValue = PifLongVal("field containing a long integer")
RetVal = IValue
```

PifGetOpenSessionTime()

Internal Basic syntax

Function PifGetOpenSessionTime() As Date

Description

This function returns the date and time of the start of the session.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

PifGetParamValue()

Internal Basic syntax

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

Description

This function is available for the AssetManagement, Database and ServiceCenter / Service Management connectors, only when the connector is in consumption mode.

This function retrieves the value stored in the 'Value' attribute associated with the 'Name' attribute whose value is equal to 'strParamName' of the collection identified by the 'strPath' relative path.

Input parameters

strParamName: Value stored in the 'Name' attribute of the PifParameters collection.

strDefaultValue: Default value stored in the 'Value' attribute of the PifParameters collection. If no value is populated, the default value is an empty string.

strPath: Relative path for the data node.

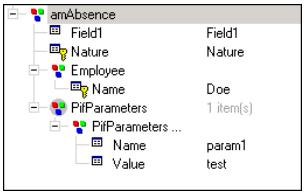
Use the ".." syntax to navigate within the mapping tree structure

Output parameters

The function returns the value as a string.

Example

The amAbsence structure contains the PifParameters collection.



The values of the amAbsence structure's child items are as follows:

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

The reconciliation script (in update mode) that calls this function is carried by the Field1 field.

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

In this example, the path (value of 'strpath') is not defined. The default value of the relative path is used (..PifParameters).

When the scenario is executed for the first time, an absence is created for employee Doe and the value for field 1 (Field) must be Field 1. When the scenario is executed for the second time, the parameter name (Name) of the PifParameters collection that is populated in the function does not exist (param2). Consequently, the specified default value is used. The value 'notest' which is set in the function is not used. In this case, the function returns the value 'notest'.

Notes

You must add the PifParameters collection to the structure or collection for this function to operate correctly.

The PifParameters collection is available for each structure or collection of the document type.

PifGetStringFromBlob()

Internal Basic syntax

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

Description

This function retrieves the value from a blob and defined by the strPath parameter and converts it to a string.

Input parameters

strPath: Path for the data node to convert.

Use the ".." syntax to navigate within the mapping tree structure

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into a string.

Example

```
Dim StrValue As String
StrValue = PifGetStringFromBlob("field containing a blob")
RetVal = StrValue
```

PifGetStringVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to a string.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value that is converted into the string format.

Example

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

The syntax to use to retrieve the value of a field for a collection item is as follows:

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

Where c is field, b is collection, and index is the number to target.

For example, for the Name field of the amComputer.LogicalDrives collection:

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

This syntax retrieves the value for the Name of the third logical disk drive (LogicalDrive(3)) in the collection.

PifGetVariantVal()

Internal Basic syntax

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

Description

This function retrieves the value defined by the strPath parameter and converts it to a variant.

Input parameters

strPath: Path for the data node to convert.

bCkeckNodeExists: This function is used to check that the data node exists in the data that is returned.

Output parameters

The function returns the value converted into a variant.

Example

```
strTest = RetVal = PifGetVariantVal("field containing a variant")
```

PifIgnoreCollectionMapping()

Internal Basic syntax

Function PifIgnoreCollectionMapping(strMsg As String) As Long

Description

This function enables you to ignore a collection, *only in a collection-to-collection mapping*.

As a reminder, the **PifIgnoreNodeMapping()** function simply enables you to ignore the current element of a collection.

An information message, contained in the **strMsg** parameter can be sent to the log file.

Input parameters

- **strMsg**: Optional parameter. Character string sent to the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

In the following example, all the elements of the collection are ignored if one of the elements has a *quantity* node whose value is set to '0':

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

For comparison with the **PifIgnoreNodeMapping()** function, in the following example, only those elements of the collection with a *quantity* node whose value is set to '0' are ignored:

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

PifIgnoreDocumentMapping()

Internal Basic syntax

Function PifIgnoreDocumentMapping(*strMsg* As String) As Long

Description

This function enables you to ignore a document.

An information message, contained in the **strMsg** parameter, can be sent to the log.

The produced document is logged in the document log but is not transmitted to the next connector.

Input parameters

- **strMsg**: Optional parameter. Character string sent to the log.

Output parameters

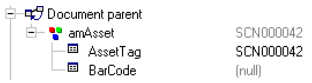
- 0: Normal execution.
- Other than zero: Error code.

Example

For example, if you want to ignore documents for which the **BarCode** field is empty, but still want to log the value of the **AssetTag** field in the tracking lines, the following script can be used on the **BarCode** node:

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

If this script is used on the document below:



the following document appears in the document log (it is not transmitted to the next connector) :

An information message is also logged in the tracking lines.

PifIgnoreDocumentReconc()

Internal Basic syntax

Function PifIgnoreDocumentReconc(*strMsg* As String) As Long

Description

This function is used to ignore a document during a reconciliation operation. No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

 Note:

This function can only be used in non-parallelized mode.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

 Note:

This function is only available in the reconciliation scripts.

PifIgnoreNodeMapping()

Internal Basic syntax

Function PifIgnoreNodeMapping(strMsg As String) As Long

Description

This function enables you to skip a node in a document.

An information message, contained in the **strMsg** parameter, can be sent to the log.

> **Warning:**
>
> This function cannot be used on the root node of a document. To ignore a full document, use the **PifIgnoreDocumentMapping** function.

Input parameters

- **strMsg**: Optional parameter. Character string sent to the log.

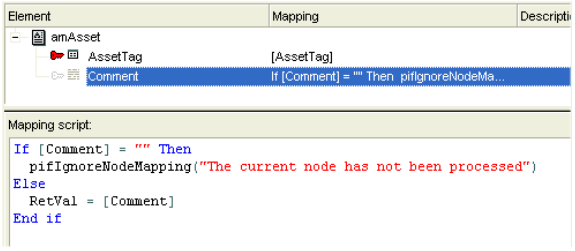
Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

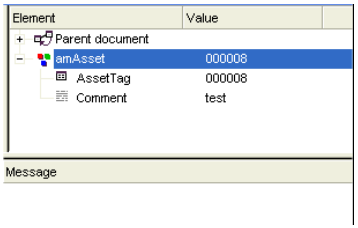
The following script ignores the current node and logs a message if the **[Comment]** field is empty.

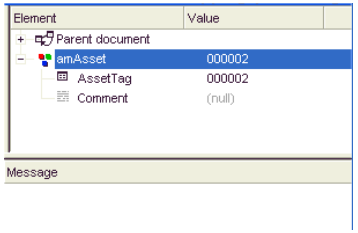
```
If [Comment] = "" Then
PifIgnoreNodeMapping("The current node was not processed")
Else
RetVal = [Comment]
End If
```



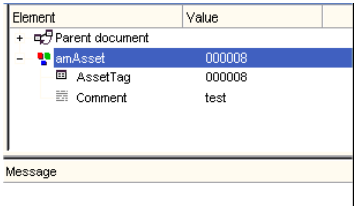
For example, if the mapping source document is the following:

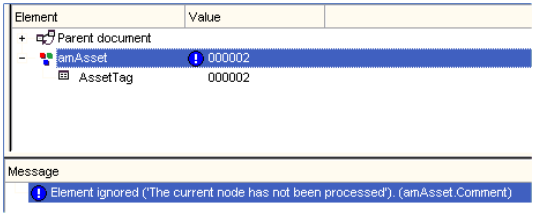
If the **PifIgnoreNodeMapping()** function is not used, the following documents can be produced:





If the **PifIgnoreNodeMapping()** is used, the following documents will be produced instead:





Notes

Note:

Since the node is not processed, the message sent to the log is written to the parent of the node that is skipped.

PifIgnoreNodeReconc()

Internal Basic syntax

Function PifIgnoreNodeReconc(strMsg As String) As Long

Description

This function is used to ignore the current node (sub-document, collection, structure, etc.) during a reconciliation operation. No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

Note:

This function is only available in Connect-It reconciliation scripts.

PifIgnoreSubDocumentReconc()

Internal Basic syntax

Function PifIgnoreSubDocumentReconc(strMsg As String) As Long

Description

This function applies when a field type element is selected. It is used in a reconciliation operation to ignore all elements on the same level as the field for which the function is applied as well as all sub-elements (links, structures, collections). No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

Note:

This function is only available in the Connect-It reconciliation scripts when a field is selected (does not apply for a link, structure or collection type element).

For a given document type whose format is:

```
Struct1
-Field1
-Field1b
-Struct2
C-Field2
```

Field 1b, structure 2 and field 2 are ignored.

PifIsInMap()

Internal Basic syntax

Function PifIsInMap(*strKey* As String, *strMaptable* As String, *bCaseSensitive* As Long) As Long

Description

This function tests if a keyword is in a maptable. The search can be made case sensitive.

Input parameters

- **strKey**: Keyword.
- **strMaptable**: Name of the maptable being searched.
- - **0**:Case insensitive.
  - **1**:Case sensitive.

  bCaseSensitive: This parameter specifies whether the search is case sensitive.

Output parameters

- 0: Keyword not found.
- 1: Keyword found.

Example

```
If PifIsInMap("CAT_PC", "MainAsset") Then
RetVal = 1
Else
RetVal = 0
End If
```

PifLogInfoMsg()

Internal Basic syntax

Function PifLogInfoMsg(*strMsg* As String) As Long

Description

Sends an information message, contained in the ***strMsg*** parameter, to the log.

Input parameters

- ***strMsg***: Character string containing the message to be sent to the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim strBrand As String
strBrand = [DeviceBrand]
If strBrand = "" Then
PifLogInfoMsg(PifStrVal("BRAND_UNREGISTERED"))
RetVal = PifStrVal("BRAND_UNKNOWN")
Else
RetVal = strBrand
End If
```

PifLogWarningMsg()

Internal Basic syntax

Function PifLogWarningMsg(*strMsg* As String) As Long

Description

Sends a warning message, contained in the **strMsg** parameter, to the log.

Input parameters

- **strMsg**: Character string containing the warning message to be sent to the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
If [Brand] = "" Then
PifLogWarningMsg("Unknown brand")
Else
RetVal = [Brand]
End if
```

PifMapValue()

Internal Basic syntax

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

Returns the value of an element in a maptable. The element is uniquely identified using the **strKey**, **strMapTable**, and **iPos** parameters.

The search can be made case sensitive.

Input parameters

- **strKey**: Keyword identifying the line in the maptable containing the element.

- **strMaptable**: Name of the maptable being searched.
- **iPos**: Number of the column containing the element whose value you want to recover.

  Note:

  This parameter can be any positive number; 0 represents the first column.

- **strDefault**: Default value returned if the element is not found.
- **bCaseSensitive**: This parameter specifies whether the search is case sensitive or not.
  - **0**: Case insensitive.
  - **1**: Case sensitive (default value).
- **bLogErrIfOutOfRange**: Depending on this parameter, en error is logged (=TRUE) or not (=FALSE) in the event log when the column number (**iPos** parameter) is either negative or greater than the number of columns in the maptable.

  Note:

  By default, this parameter is set to TRUE.

- **bLogNewEntries**: This parameter creates the maptable for keys found and not previously declared in a maptable in a new file adjoining the scenario. This file has the same name as the scenario but is prefixed by _NewMapKeys.mpt.

  Note:

  Maptables concerning the scenario are not updated automatically. To update maptables, edit the file that was just created and copy/paste the new information in the maptable file.

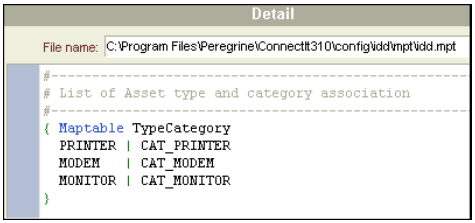
  - **0** (default value): The automatic creation of maptable files is disabled.
  - **1**: The automatic creation of maptable files is enabled.

Output parameters

The function returns the string found or the default string (which is contained in the **strDefault** parameter) if the element is not found.

Example

For example, consider the following maptable:



The following script:

```
pifMapValue("PRINTER", "TypeCategory", 1, "")
```

returns **CAT_PRINTER**.

The following script:

```
pifMapValue("Printer", "TypeCategory", 1, "")
```

returns the default value. The search being case-sensitive, the searched element is not found.

The following script:

```
pifMapValueContaining("Print", "TypeCategory", 1, 1, 1 "")
```

returns the default value, saves an error to the event log, and creates a maptable file.

Notes

Note:

This function correctly handles the following wildcard characters:

- ?: replaces any character.
- *: replaces any number of characters.

PifMapValueContaining()

Internal Basic syntax

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

This function returns the value of an element in a maptable. The element is uniquely identified using the **strKey**, **strMapTable**, and **iPos** parameters.

The search can be made case sensitive.

The difference with the **PifMapValue** function is that the **strKey** parameter can be a subset of the keyword being searched.

For example, if **strKey** contains "Monitor", the element with keyword "Cat_Monitor" will be found.

Input parameters

- **strKey**: Keyword or subset of keyword identifying the line in the maptable containing the element.
- **strMaptable**: Name of the maptable being searched.
- **iPos**: Number of the column containing the element whose value you want to recover.

  Note:

  This parameter can be any positive number; 0 represents the first column.

- **strDefault**: Default value returned if the element is not found.
- - **0**: Case insensitive.
  - **1**: Case sensitive (default value).

  bCaseSensitive: This parameter specifies whether the search is case sensitive.
- **bLogErrIfOutOfRange** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when the column number (**iPos** parameter) is either negative or greater than the number of columns in the maptable.

Note:

By default, this parameter is set to TRUE.

- **bLogNewEntries**: This parameter creates the maptable for keys found and not previously declared in a maptable in a new file adjoining the scenario. This file has the same name as the scenario but is prefixed by _NewMapKeys.mpt.

Note:

Maptables concerning the scenario are not updated automatically. To update maptables, edit the file that was just created and copy/paste the new information in the maptable file.

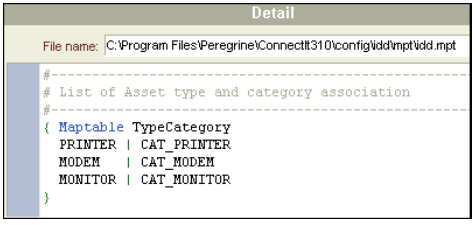
  - **0** (default value): The automatic creation of maptable files is disabled.
  - **1**: The automatic creation of maptable files is enabled.

Output parameters

The function returns the string found or the default string (which is contained in the **strDefault** parameter) if the element is not found.

Example

For example, with the following maptable:



The following script:

```
pifMapValueContaining("PRINT", "TypeCategory", 1, "")
```

returns **CAT_PRINTER**.

Note:

If several keys in the maptable match the **StrKey** parameter, then the function will return first key encountered.

The following script:

```
pifMapValueContaining("Print", "TypeCategory", 1, "")
```

returns the default value. As the search is case-sensitive, the element is not found.

The following script:

```
pifMapValueContaining("Print", "TypeCategory", 1, 1, 1 "")
```

returns the default value, saves an error in the event log and creates a maptable file.

Notes

Note:

This function correctly handles the following wildcard characters:

- ?: replaces any character.
- *: replaces any number of characters.

PifMapValueContainingEx()

Internal Basic syntax

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

This function returns the value of an element in a maptable. The element is uniquely identified by these parameters: **strKey**, **strMapTable**, **strColName**.

The search for the element can be made case sensitive.

As opposed to the **PifMapContaining** function, the **strColName** parameter defines the column name instead of its position.

This function is useful when:

- The maptable was created from a format (the column names correspond to the element names of the format)
- The column names were defined manually in the maptables using the maptable editor.

Input parameters

- **strKey**: Expression containing one of the keys defined in the maptable.
- **strMaptable**: Name of the maptable being searched.
- **strColName**: Name of the column containing the element whose value you want to retrieve.
- **strDefault**: Default value returned if the element is not found.
- - **0**: The search is not case sensitive.
  - **1** (default value): The search is case sensitive.

  bCaseSensitive: This parameter specifies whether the search is case sensitive or not.
- **bLogErrIfOutOfRange**: Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when the column number (**iPos** parameter) is either negative or greater than the number of columns in the maptable.

   Note:

  By default, this parameter is set to TRUE.
- **bLogNewEntries**: This parameter creates the maptable for keys found and not previously declared in a maptable in a new file adjoining the scenario. This file has the same name as the scenario but is prefixed by _NewMapKeys.mpt.

   Note:

  Maptables concerning the scenario are not updated automatically. To update maptables, edit the file that was just created and copy/paste the new information in the maptable file.

  - **0** (default value): The automatic creation of maptable files is disabled.
  - **1**: The automatic creation of maptable files is enabled.

Output parameters

The function returns the string found or the default string (which is contained in the **strDefault** parameter) if the element is not found.

Notes

Note:

This function correctly handles the following wildcard characters:

- ? : Corresponds to any character.
- * : Corresponds to any number of characters.

If no maptable is created from a format or manually using the maptable editor, use the **PifMapValueContaining** or **PifMapValue** functions.

PifMapValueEx()

Internal Basic syntax

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

This function returns the value of an element in a maptable. The element is uniquely identified by these parameters: **strKey**, **strMapTable**, **strColName**. The difference of this function as compared to the **PIFMAPVALUE** function is that the **strColName** parameter specifies the column name instead of its number.

The search for the element can be made case sensitive.

This function is useful when:

- The maptable was created from a format (the column names correspond to the element names of the format)
- The column names were defined manually in the maptables using the maptable editor.

Input parameters

- **strKey**: Keyword that is used to identify the line in the maptable that contains the element.
- **strMaptable**: Name of the maptable being searched.
- **strColName**: Name of the column containing the element whose value you want to retrieve.
- **strDefault**: Default value returned if the element is not found.
- **bCaseSensitive**: This parameter specifies whether the search is case sensitive or not.
  - **0**: The search is not case sensitive.
  - **1** (default value): The search is case sensitive.
- **bLogErrIfOutOfRange**: Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when the column number (**iPos** parameter) is either negative or greater than the number of columns in the maptable.

   Note:

  By default, this parameter is set to TRUE.

- **bLogNewEntries**: This parameter creates the maptable for keys found and not previously declared in a maptable in a new file adjoining the scenario. This file has the same name as the scenario but is prefixed by _NewMapKeys.mpt.

   Note:

  Maptables concerning the scenario are not updated automatically. To update maptables, edit the file that was just created and copy/paste the new information in the maptable file.

  - **0** (default value): The automatic creation of maptable files is disabled.
  - **1**: The automatic creation of maptable files is enabled.

Output parameters

The function returns the string found or the default string (which is contained in the **strDefault** parameter) if the element is not found.

Notes

> Note:
>
> This function correctly handles the following wildcard characters:
> - ? : Corresponds to any character.
> - * : Corresponds to any number of characters.

If no maptable is created from a format or manually using the maptable editor, use the **PifMapValueContaining** or **PifMapValue** functions.

PifNewQueryFromFmtName()

Internal Basic syntax

Function PifNewQueryFromFmtName(strCntrName As String, strFmtName As String, strLayer As String) As Long

Description

This function creates a query on the document type defined beforehand in the list of documents produced by a resource.

Input parameters

- **strCntrName**: This parameter contains the name of the resource (on which the query is performed).
- **strFmtName**: This parameter contains the identifier of the document type (defined beforehand as a produced document type).
- **strLayer**: This parameter is used to define a production directive for a produced document type.

  The production directive can be:
  - A WHERE clause that uses the correct syntax for the connector
  - An XML description that defines production directives that are applied for the connector (for example, an Order By clause). The XML syntax of the production directives is described in the **PifNewQueryFromXML** documentation.

Note:

If the document type **strFmtName** already contains production directives, they will be merged with those provided in the **strLayer** parameter. In the event that the two parameters provide a different value for the same directive, the value defined in **strLayer** will prevail.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example shows a simple Where clause for the **strLayer** parameter:

```
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)
```

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifQueryClose() [page 156]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifNewQueryFromXml()

Internal Basic syntax

Function PifNewQueryFromXml(strCntrName As String, strQuery As String, strLayer As String) As Long

Description

This function creates a query for a resource. The document type must be fully defined in XML by the **strQuery** parameter. >The processing (AQL query clause) is defined in XML by the **strLayer** parameter.

Input parameters

- **strCntrName**: This parameter contains the name of the resource (on which the query is performed).
- **strQuery**: This parameter contains an XML document that defines the document type (attributes, structures, collections) on which the query is performed.
- **strLayer**: This parameter contains an XML document that defines the production directives (Where clause, Order By clause, ...) for the query.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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>"
```

Notes

In this example, we build an XML document that specifies both the document to produce and the clauses of the query executed. As for all XML documents, it must be valid. Reserved characters (for example <, &...) must be escaped using the GetXmlElementValue() [page 85] and GetXmlAttributeValue() [page 85] functions.

Note:

When **strLayer** contains a simple AQL Where clause, it is not necessary to use GetXmlElementValue() [page 85].

The preceding description assumes that you know the type of document to be produced and that the query contains a simple WHERE clause. The example below shows how to use the function without knowing the type of document to produce. It also shows how to use other AQL clauses.

```
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:

In this example, we build an XML document that specifies both the document to produce and the clauses of the query executed. As for all XML documents, it must be valid. Reserved characters (for example <,&...) must be replaced with the corresponding entities (<,&...).

See also:

- PifQueryClose() [page 156]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifNodeExists()

Internal Basic syntax

Function PifNodeExists(strPath As String) As Long

Description

Tests whether a node, identified by its full access path, exists in a produced document.

Input parameters

- **strPath**: Full path of the node concerned by the operation.

Output parameters

Returns either of the following values:

- **0**:if the node does not exist.
- **1**:if the node exists

Example

```
If PifNodeExists("Hardware.Peripherials.Printer") = 0 Then
PifIgnoreNodeMapping
End If
```

PifOpenODBCDatabase()

Internal Basic syntax

Function PifOpenODBCDatabase(strDSN As String, strLogin As String, strPwd As String) As Long

Description

This function opens a connection to an ODBC database.

Input parameters

- **strDSN**: Name of the ODBC data source for the connection.
- **strLogin**: Login used to connect to the ODBC database.
- **strPwd**: Password associated with the login (**strLogin**) used to connect to the ODBC database.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
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)
```

Notes

> Note:
>
> If there is already a connection to the data source, the function will not create a second one.

PifQueryClose()

Internal Basic syntax

Function PifQueryClose(lQueryHandle As Long) As Long

Description

This function closes the query and frees the internal resources used by the query.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
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)
```

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifQueryGetDateVal()

Internal Basic syntax

Function PifQueryGetDateVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Date

Description

This function returns the value (as a Date) of a node of the current document. The current document is the one on which the query cursor has been set using the **PifQueryNext()** function.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.
- **strPath**: This parameter contains the path of the node of the current document for which you want to recover the value.
- **bPathMustExist** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when no value can be retrieved for the path specified in the **strPath** parameter.

Output parameters

Value of the identified node

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryClose() [page 156]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifQueryGetDoubleVal()

Internal Basic syntax

Function PifQueryGetDoubleVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Double

Description

This function returns the value (as a Double) of a node of the current document. The current document is the one on which the query cursor has been set using the **PifQueryNext()** function.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.
- **strPath**: This parameter contains the path of the node of the current document for which you want to recover the value.
- **bPathMustExist** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when no value can be retrieved for the path specified in the **strPath** parameter.

Output parameters

Value of the identified node

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryClose() [page 156]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifQueryGetIntVal()

Internal Basic syntax

Function PifQueryGetIntVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Long

Description

This function returns the value (as an Integer) of a node of the current document. The current document is the one on which the query cursor has been set using the **PifQueryNext()** function.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.
- **strPath**: This parameter contains the path of the node of the current document for which you want to recover the value.
- **bPathMustExist** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when no value can be retrieved for the path specified in the **strPath** parameter.

Output parameters

Value of the identified node

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryClose() [page 156]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifQueryGetLongVal()

Internal Basic syntax

Function PifQueryGetLongVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As Long

Description

This function returns the value (as a Long) of a node of the current document. The current document is the one on which the query cursor has been set using the **PifQueryNext()** function.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.
- **strPath**: This parameter contains the path of the node of the current document for which you want to recover the value.
- **bPathMustExist** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when no value can be retrieved for the path specified in the **strPath** parameter.

Output parameters

Value of the identified node

Example

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

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryClose() [page 156]
- PifQueryGetStringVal() [page 162]

PifQueryGetStringVal()

Internal Basic syntax

Function PifQueryGetStringVal(lQueryHandle As Long, strPath As String, bPathMustExist As Long) As String

Description

This function returns the value (as a string) of a node of the current document. The current document is the one on which the query cursor has been set using the **PifQueryNext()** function.

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.
- **strPath**: This parameter contains the path of the node of the current document for which you want to recover the value.
- **bPathMustExist** : Depending on this parameter, an error is logged (=TRUE) or not (=FALSE) in the event log when no value can be retrieved for the path specified in the **strPath** parameter.

Output parameters

Value of the identified node

Notes

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryNext() [page 163]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryClose() [page 156]

PifQueryNext()

Internal Basic syntax

Function PifQueryNext(lQueryHandle As Long) As Long

Description

This function set the query cursor on the next result. The cursor is not set on the first document after calling the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions. The user must call the **PifQueryNext()** function to access the values of the current document with one of the following functions:

- **PifQueryGetStringVal()**
- **PifQueryGetDateVal()**
- **PifQueryGetDoubleVal()**
- **PifQueryGetLongVal()**
- **PifQueryGetIntVal()**

Input parameters

- **lQueryHandle**: This parameter contains a handle of the query created using the **PifNewQueryFromFmtName()** or **PifNewQueryFromXml()** functions.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
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)
```

Notes

The function returns an error when there is no more data to browse (error code -2003).

See also:

- PifNewQueryFromXml() [page 152]
- PifNewQueryFromFmtName() [page 150]
- PifQueryClose() [page 156]
- PifQueryGetDateVal() [page 157]
- PifQueryGetDoubleVal() [page 158]
- PifQueryGetIntVal() [page 159]
- PifQueryGetLongVal() [page 160]
- PifQueryGetStringVal() [page 162]

PifRejectCollectionMapping()

Internal Basic syntax

Function PifRejectCollectionMapping(strMsg As String) As Long

Description

This function enables you to reject all the elements of a collection, *only in a collection-to-collection mapping*.

As a reminder, the **PifRejectNodeMapping()** function simply enables you to reject the current node of a collection.

A message, contained in the **strMsg** parameter is sent to the log file.

> **Note:**
> Using this function is equivalent to a partial rejection of the document. Partial rejections can be recovered in the process reports of the mapping box.

Input parameters

- **strMsg**: This optional parameter contains the message to be written to the log file.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

In the following example, all the elements of the collection are rejected if one of the elements has a *quantity* node whose value is set to '-1':

```
if [root.item(pifGetInstance).quantity] = -1 then
pifRejectCollectionMapping("invalid quantity")
end if
```

For comparison with the **PifRejectNodeMapping()** function, in the following example, only those elements of the collection with a *quantity* node whose value is set to '-1' are rejected:

```
if [root.item(pifGetInstance).quantity] = -1 then
pifRejectNodeMapping
end if
```

Notes

See also:

- PifRejectNodeMapping() [page 167]
- PifRejectDocumentMapping() [page 166]

PifRejectDocumentMapping()

Internal Basic syntax

Function PifRejectDocumentMapping(strMsg As String) As Long

Description

This function enables you to reject a document.

An information message, contained in the **strMsg** parameter, can be sent to the log.

The document is not transmitted to the next connector.

Input parameters

- **strMsg**: Optional parameter. Character string sent to the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim strNetAddress As String
strNetAddress = [Hardware.TCPIP.PhysicalAddress]

If strNetAddress = "" Then
PifRejectDocumentMapping("Document rejected: missing MAC address")
Else
RetVal = strNetAddress
End If
```

PifRejectDocumentReconc()

Internal Basic syntax

Function PifRejectDocumentReconc(strMsg As String) As Long

Description

This function is used to reject a document during a reconciliation operation. No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

 Note:

This function can only be used in non-parallelized mode.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

 Note:

This function is only available in the Connect-It reconciliation scripts.

See also:

- PifRejectNodeReconc() [page 168]
- PifRejectSubDocumentReconc() [page 169]

PifRejectNodeMapping()

Internal Basic syntax

Function PifRejectNodeMapping(strMsg As String) As Long

Description

This function enables you to reject the current node of a document.

An information message, contained in the **strMsg** parameter, can be sent to the log.

Note:

Using this function is equivalent to a partial rejection of the document. Partial rejections can be recovered in the process reports of the mapping box.

Input parameters

- **strMsg**: Optional parameter. Character string sent to the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
If [Location.Name] = "" Then
PifRejectNodeMapping(PifStrVal("UNKNOWN_LOCATION"))
End If
```

Notes

Warning:

This function cannot be used on the root node of a document. To reject a full document, use the **PifRejectDocumentMapping** function.

PifRejectNodeReconc()

Internal Basic syntax

Function PifRejectNodeReconc(strMsg As String) As Long

Description

This function is used to reject the current node (sub-document, collection, structure, etc.) during a reconciliation operation. No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

 Note:

This function is only available in Connect-It reconciliation scripts.

PifRejectSubDocumentReconc()

Internal Basic syntax

Function PifRejectSubDocumentReconc(strMsg As String) As Long

Description

This function is used to reject a sub-document during a reconciliation operation. No insertion or update is performed. A message, stored in the **strMsg** parameter, can be displayed in the log.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

> Note:
> This function is only available in the reconciliation scripts.

PifScenarioPath()

Internal Basic syntax

Function PifScenarioPath() As String

Description

This function returns the full path of a scenario file. If the scenario has not yet been saved the function returns an empty value.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strFile
strFile = pifScenarioPath & "Format.xml"

Open strFile For Output As #1
Print #1, pifXmlDump
Close #1
```

PifSetDateVal()

Internal Basic syntax

Function PifSetDateVal(strPath As String, dttVal As Date) As Long

Description

This function enables you to set the value of a node in the target document. If the node does not exist, the function creates it.

Input parameters

- **strPath**: This parameter contains the full path of the node concerned by the operation.
- **dttVal**: This parameter contains the value (date) that you want to assign to the node.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim dtCurrent as Date
Dim lRet as Long
dtCurrent = Date()
lRet = PifSetDateVal("ValueDate", dtCurrent)
```

Notes

See also:

- PifSetDoubleVal() [page 172]
- PifSetLongVal() [page 173]
- PifSetNullVal() [page 174]
- PifSetStringVal() [page 177]

PifSetDoubleVal()

Internal Basic syntax

Function PifSetDoubleVal(strPath As String, dVal As Double) As Long

Description

This function enables you to set the value of a node in the target document. If the node does not exist, the function creates it.

Input parameters

- **strPath**: This parameter contains the full path of the node concerned by the operation.
- **dVal**: This parameter contains the value (double) that you want to assign to the node.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim d as Double
Dim lRet as Long
d = 2.5
lRet = PifSetDoubleVal("ValueDouble", d)
```

Notes

See also:

- PifSetDateVal() [page 171]
- PifSetLongVal() [page 173]
- PifSetNullVal() [page 174]
- PifSetStringVal() [page 177]

PifSetLongVal()

Internal Basic syntax

Function PifSetLongVal(strPath As String, lVal As Long) As Long

Description

This function enables you to set the value of a node in the target document. If the node does not exist, the function creates it.

Input parameters

- **strPath**: This parameter contains the full path of the node concerned by the operation.
- **lVal**: This parameter contains the value (long integer) that you want to assign to the node.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim l as Long
Dim lRet as Long
l = 2
lRet = PifSetLongVal("ValueLong", l)
```

Notes

See also:

- PifSetDoubleVal() [page 172]
- PifSetDateVal() [page 171]
- PifSetNullVal() [page 174]
- PifSetStringVal() [page 177]

PifSetNullVal()

Internal Basic syntax

Function PifSetNullVal(strPath As String) As Long

Description

This function enables you to populate a node in the target document with the value 'NULL'. If the node does not exist, the function will create it.

Input parameters

- **strPath**: This parameter contains the full name of the node concerned by the operation. If this parameter is omitted or empty, the current node is selected.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
If [Name] = "" Then
PifSetNullVal("FirstName")
End if
```

Notes

See also:

- PifSetDoubleVal() [page 172]
- PifSetLongVal() [page 173]
- PifSetDateVal() [page 171]
- PifSetStringVal() [page 177]

PifSetParamValue()

Internal Basic syntax

Function PifSetParamValue(*strParamName* As String, *strValue* As String, *strPath* As String)

Description

This function is available for the AssetManagement, Database and ServiceCenter / Service Management connectors, only when the connector is in consumption mode.

This function is used to define a text type value which can be accessed in a mapping script or a reconciliation script.

Input parameters

strParamName: Value stored in the 'Name' attribute of the PifParameters collection.

strValue: Value stored in the 'Value' attribute of the PifParameters collection.

strPath: Relative path for the data node.

Collection nodes must have been created and a value set for the 'value' attribute.

Use the ".." syntax to navigate within the mapping tree structure

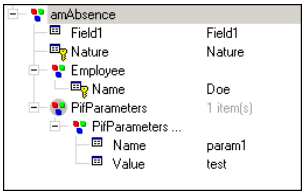
Output parameters

The function returns the value as a string.

When the scenario is executed for the second time, field 1 keeps its value (because vOldVal=vNewVal='Field1'). The reconciliation script changes the 'param1' parameter by assigning the value 'newvalue' to it (the initial value was 'test').

Example

The amAbsence structure contains the PifParameters collection.



The values of the amAbsence structure's child items are as follows:

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

The reconciliation script (in update mode) that calls this function is carried by the Nature field.

```
call PifSetParamValue("param1", "newValue")
RetVal = vNewVal
```

When the scenario is executed for the first time, an absence is created for employee Doe. The value for field 1 is 'Field 1' and the value for the Nature field is 'nature'.

Notes

You must add the PifParameters collection to the structure or collection for this function to operate correctly.

The PifParameters collection is available for each structure or collection of the document type, in consumption mode.

PifSetPendingDocument()

Internal Basic syntax

Function PifSetPendingDocument(strMsg As String) As Long

Description

This function is used to instruct a document to wait during a reconciliation operation. The document is not inserted or updated. A message, stored in the **strMsg** parameter, can be displayed in the log.

Note:

This function can only be used in non-parallelized mode.

Input parameters

- **strMsg** : This parameter contains the message displayed in the log.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
If vNewVal >= vOldVal Then
RetVal = vNewVal
Else
RetVal = vOld
PifSetPendingDocument(PifStrVal("RECONC_PROPOSAL_NOT_VALIDATED"))
End If
```

Notes

Note:

This function is only available in the reconciliation scripts.

PifSetStringVal()

Internal Basic syntax

Function PifSetStringVal(strPath As String, strVal As String) As Long

Description

This function enables you to set the value of a node in the target document. If the node does not exist, the function creates it.

Input parameters

- **strPath**: This parameter contains the full path of the node concerned by the operation.
- **strVal**: This parameter contains the value (character string) that you want to assign to a node.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
Dim str as String
Dim lRet as Long
str = "100.10.1.1"
lRet = PifSetStringVal("ipaddress", str)
```

Note:

You must retrieve the return code for the function. Failure to do so will result in a compilation error.

The syntax to use to set the value of a field for a collection item is as follows:

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

Where c is field, b is collection, and index is the number to target.

For example, for the Name field of the amComputer.LogicalDrives collection:

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

This syntax is used to set the value for the Name of the third logical disk drive (LogicalDrive(3)) in the collection.

Notes

See also:

- PifSetDoubleVal() [page 172]

- PifSetLongVal() [page 173]
- PifSetNullVal() [page 174]
- PifSetDateVal() [page 171]

PifStrVal()

Internal Basic syntax

Function PifStrVal(strID As String) As String

Description

Returns the character string associated with the identifier contained in the **strId** parameter.

Input parameters

- **strID**: Identifier of the character string to recover.

Output parameters

If the identifier is not found, the function returns an empty string and writes an error in the Connect-It log. If the identifier is found, the function returns its associated character string.

Example

```
If [DeviceType] = "" Then
RetVal = PifStrVal("BRAND_UNKNOWN")
End If
```

PifUserFmtStrToVar()

Internal Basic syntax

Function PifUserFmtStrToVar(strData As String, strUserFmtName As String) As Variant

Description

This function processes a character string according to a format predefined using a wizard in the graphical interface of Connect-It, and returns a Date or Number type number according to the nature of the predefined format.

Input parameters

- **strData**: This parameter contains the string to be processed.
- **strUserFmtName**: This parameter contains the name of the predefined format.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

If we consider the following two predefined formats:

- origin = "yyyy'-'mm'-'dd"
- destination = "'dddd' 'dd' 'mmmm' 'yyyy'"

Then the script:

```
Dim dTmp as Variant
dTmp=PifUserFmtVarToStr([date_modified],"origin")
RetVal = PifUserFmtStrToVar(dTmp,"destination")
```

Returns for [date_modified]= 2001-05-30 the value "Thursday, May 30, 2001"

Notes

Note:

For further information on the predefined formats, refer to the Connect-It User's Guide

See also :

- PifUserFmtVarToStr() [page 181]

PifUserFmtVarToStr()

Internal Basic syntax

Function PifUserFmtVarToStr(vData As Variant, strUserFmtName As String) As String

Description

This function processes a variant according to a predefined format and returns a character string.

Input parameters

- **vData**: This parameter contains the variant processed by the function.
- **strUserFmtName**: This parameter contains the name of the predefined format.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

If we consider the following two predefined formats:

- origin = "yyyy'-'mm'-'dd"
- destination = "'dddd' 'dd' 'mmmm' 'yyyy'"

Then the script:

```
Dim dTmp as Variant
dTmp=PifUserFmtVarToStr([date_modified],"origin")
RetVal = PifUserFmtStrToVar(dTmp,"destination")
```

Returns for [date_modified]= 2001-05-30 the value "Thursday, May 30, 2001"

Notes

Note:

For further information on predefined formats, refer to the Connect-It User's Guide

See also :

- PifUserFmtStrToVar() [page 179]

PifWriteBlobInFile()

Internal Basic syntax

Function PifWriteBlobInFile(strPath As String, strFileName As String) As Long

Description

This function enables you to write a binary element (blob) to a file. If the file does not exist, it is created by the function. If the file already exists, it is overwritten by the function.

Input parameters

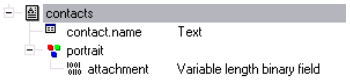
- **strPath** : This parameter contains the full path of the binary element (blob) in the source document.
- **strFileName**: This parameter contains the full path of the file in which the binary object is saved.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

The screen capture below shows a source document containing the name and the picture of an employee:



The following script saves the picture of each employee in the c:\bitmap folder. The filename is the name of the employee.

```
Dim lErr As Long
Dim strFileName As String

strFileName = "C:\bitmap\" & ['contact.name'] & ".bmp"
lErr = PifWriteBlobInFile("portrait.attachment", strFileName)
```

Note:

- The **contact.name** element contains a dot (."). It is therefore mandatory to surround it by quotes to reference it (['contact.name']).
- When a function returns an error code, a variable should always be assigned to the function return code (as in : **lErr = PifWriteBlobInFile()**). If this is not the case, the script is not valid.

Notes

The function returns an error in the following cases:

- the path of the file is not valid,
- the path of the source element does not exist in the source document,
- the source element is not a binary element.

PifXMLDump()

Internal Basic syntax

Function PifXMLDump(strPath As String) As String

Description

This function enables you to dump the content of an element (and all its sub-elements) of the source document in XML format, inside a string.

Input parameters

- **strPath**: This parameter contains the full path of the element of the source document concerned by the operation.

Output parameters

In case of error, a message is written to the Connect-It log.

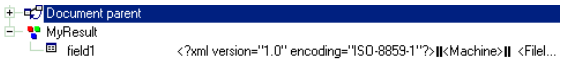
Example

This first example describes how to save the content of the element below with the **PifXMLDump()** function.

The following script, associated to the "field1" field, is used:

```
RetVal = PifXMLDump("")
```

The screen capture below shows the result of the script execution.



"field1" contains the following XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Machine>
<FileInfo>
<FileName>F:\ConnectIt4016\datakit\idd\Frankfurt01.fsf</FileName>
</FileInfo>
</Machine>
```

To save only the structure of the FileInfo element, the following script can be used:

```
RetVal = PifXMLDump("FileInfo")
```

The XML document contained in the field will then be:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FileInfo>
<FileName>F:\ConnectIt4016\datakit\idd\Frankfurt01.fsf</FileName>
</FileInfo>
```

PMT()

Internal Basic syntax

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

Description

This function returns the amount of an annuity based on constant and periodic payments, and at a constant interest rate.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

- **iNper**: This parameter contains the total number of dates of payment for the financial operation.
- **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.
- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

- The **Rate** and **Nper** parameters must be calculated using payments expressed in the same units.
- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

PPMT()

Internal Basic syntax

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

Description

This function returns the amount of capital reimbursed for a given date of payment in an annuity based on constant and periodic payments and at a constant interest rate.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

- **iPer**: This parameter indicates the period for the calculation, between 1 and the value of the **Nper** parameter.
- **iNper**: This parameter contains the total number of dates of payment for the financial operation.
- **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.
- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.

- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

 Note:

- The **Rate** and **Nper** parameters must be calculated using payments expressed in the same units.
- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

PV()

Internal Basic syntax

Function PV(dblRate As Double, iNper As Long, dblPmt As Double, dblFV As Double, iType As Long) As Double

Description

This function returns the actual amount of an annuity based on constant and periodic future deadlines, and on a fixed interest rate.

Input parameters

- **dblRate**: This parameter indicates the interest rate per date of payment. For example, the rate per date of payment for a loan with a 6% annual interest rate, paid back by monthly dates of payment, would be:

```
0.06/12=0.005 or 0.5%
```

- **iNper**: This parameter contains the total number of dates of payment for the financial operation.
- **dblPmt**: This parameter indicates the amount of the payment made at each date of payment. The payment generally includes both principal and interest.
- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

- The **Rate** and **Nper** parameters must be calculated using payments expressed in the same units.
- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.

Randomize()

Internal Basic syntax

Function Randomize(lValue As Long)

Description

Initializes the random number generator.

Input parameters

- **lValue**: Optional parameter used to initialize the random-number generator of the **Rnd** function by specifying a new initial value. If this parameter is omitted , the value returned by the system clock is used as the initial value.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyNumber
Randomize
MyNumber= Int((10*Rnd)+1) :'Returns a random value between 1 and 10.
RetVal=MyNumber
```

Notes

See also:

- Rnd() [page 197]

RATE()

Internal Basic syntax

Function RATE(iNper As Long, dblPmt As Double, dblFV As Double, dblPV As Double, iType As Long, dblGuess As Double) As Double

Description

This function returns the interest rate per date of payment for an annuity.

Input parameters

- **iNper**: This parameter contains the total number of dates of payment for the financial operation.
- **dblPmt**: This parameter indicates the amount of the payment to be made at each date of payment. The payment generally includes both principal and interest.

- **dblFV**: This parameter contains the future value or the balance that you want to obtain after having paid the final date of payment. In general, and particularly when reimbursing a loan, this parameter is set to "0". In effect, once you have made all the dates of payment, the value of the loan is nil.
- **dblPV**: This parameter contains the actual value (or overall sum) for a series of payments to be made in the future.
- **iType**: This parameter indicates the payment deadline. It can have one of the following values:
  - **0** if the payments are due in arrears (i.e. at the end of the period)
  - **1** if the payments are due in advance (i.e. at the start of the period)
- **dblGuess**: This parameter contains the estimated value of the interest rate per date of payment.

Output parameters

In case of error, a message is written to the Connect-It log.

Notes

Note:

- Amounts paid (expressed in particular by the **Pmt** parameter) are represented by negative numbers. Sums received are represented by positive numbers.
- This function performs its calculation using iterations, starting with the value assigned in the **Guess** parameter. If no result is found after 20 iterations, the function fails.

RemoveRows()

Internal Basic syntax

Function RemoveRows(strList As String, strRowNames As String) As String

Description

Performs a deletion in a list of lines identified by the **strRowNames** parameter.

This function is useful when processing "ListBox" control type values. Values from this type of control are represented as arrays as described below:

- The "|" character is used as the column separator.
- The "," character is used as the line separator.
- Each line ends with a unique identifier at the right of the "=" sign.

Input parameters

- **strList**: Source string containing the values of a "ListBox" control to be processed.
- **strRowNames**: Identifiers of lines to be deleted. The identifiers are separated by commas.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=RemoveRows("a1|a2=a0,b1|b2=b0", "a0,c0") :'Returns "b1|b2=b0"
RetVal=MyStr
```

Notes

See also:

- SubList() [page 208]
- SetSubList() [page 201]
- ApplyNewVals() [page 57]

Replace()

Internal Basic syntax

Function Replace(strData As String, strOldPattern As String, strNewPattern As String, bCaseSensitive As Long) As String

Description

Replaces all occurrences of the **strOldPattern** parameter with the **strNewPattern** parameter inside the character string contained in the **strData** parameter. The search for the **strOldPattern** parameter can be made case-sensitive using the value of the **bCaseSensitive** parameter.

Input parameters

- **strData**: Character string containing the occurrences to be replaced.
- **strOldPattern**: Occurrence to find in the string contained in the **strData** parameter.
- **strNewPattern**: Text replacing each occurrence found.
- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0). By default, this parameter is set to 1.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=Replace("youmeyoumeyou", "you", "me",0) :'Returns "mememememe"
MyStr=Replace("youmeyoumeyou", "You", "me",1) :'Returns "youmeyoumeyou"
MyStr=Replace("youmeYoumeyou", "You", "me",1) :'Returns "youmememeyou"
```

Right()

Internal Basic syntax

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

Description

Returns the rights most iNumber characters of the string parameter.

Input parameters

- **strString**: Character string to process.
- **lNumber**: Number of characters to return.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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()

Internal Basic syntax

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

Description

Extracts the portion of a string to the right of the separator specified in the **strSep** parameter.

The search for the separator is performed from right to left.

The search can be made case sensitive using the **bCaseSensitive** parameter.

Input parameters

- **strFrom**: Source string to be processed.
- **strSep**: Character used as separator in the source string.
- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0).

Output parameters

In case of error, a message is written to the Connect-It log.

Example

These examples illustrate use of the **LeftPart**, **LeftPartFromRight**, **RightPart**, and **RightPartFromLeft** functions on the same string: "This_is_a_test":

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

Returns "This".

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

Returns "This_is_a".

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

Returns "test".

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

Returns "is_a_test".

RightPartFromLeft()

Internal Basic syntax

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

Description

Extracts the portion of a string to the right of the separator specified in the **strSep** parameter.

The search for the separator is performed from left to right.

The search can be made case sensitive using the **bCaseSensitive** parameter.

Input parameters

- **strFrom**: Source string to be processed.
- **strSep**: Character used as separator in the source string.

- **bCaseSensitive**: Depending on this parameter, the search is case sensitive (=1) or not (=0). By default, this parameter is set to 1.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

These examples illustrate use of the **LeftPart**, **LeftPartFromRight**, **RightPart**, and **RightPartFromLeft** functions on the same string: "This_is_a_test":

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

Returns "This".

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

Returns "This_is_a".

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

Returns "test".

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

Returns "is_a_test".

RmAllInDir()

Internal Basic syntax

Function RmAllInDir(strRmDirectory As String, bStopIfError As Long) As Long

Description

This function deletes all items (files and folders) from a folder. The folder itself is not deleted.

Input parameters

- **strRmDirectory**: This parameter contains the full path of the folder concerned by the operation.

- **bStopIfError**: If this parameter is set to 1, the delete operation is suspended if the a file or folder cannot be deleted. If this parameter is set to 0, the operation continues and moves on to the following file or folder.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
RetVal = RmAllInDir("c:\files\test", 1)
```

RmDir()

Internal Basic syntax

Function RmDir(strRmDirectory As String) As Long

Description

Removes an existing directory.

Input parameters

- **strRmDirectory**: Full path of the directory to be removed.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Example

```
RetVal = RmDir("c: mp")
```

Notes

> Note:
> The directory to be deleted must be empty. Otherwise, the function will not work.

Rnd()

Internal Basic syntax

Function Rnd(dValue As Double) As Double

Description

Returns a value containing a random number.

Input parameters

- **dValue**: Optional parameter whose value defines the mode of execution of the function:
  - Less than zero: The same number is generated each time.
  - Greater than zero: Next random number in the series.
  - Equal to zero: Last random number generated.
  - Omitted: Next random number in the series.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyNumber
Randomize
MyNumber= Int((10*Rnd)+1) :'Returns a random value between 1 and 10.
RetVal=MyNumber
```

Notes

> Note:
>
> Before calling this function, you must use the **Randomize** function, without parameters, to initialize the random number generator.

See also:

- Randomize() [page 188]

RoundValue()

Internal Basic syntax

Function RoundValue(dValue As Double, iDigits As Long) As Double

Description

This function calculates the rounding value of a number to the number of digits after the decimal point as specified by the **iDigits** parameter.

Input parameters

- **dValue**: This parameter contains the number to be rounded.
- **iDigits**: This parameter contains the number of decimal places to keep for the rounding operation.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

```
RetVal = RoundValue(1.2568, 2)
```

returns the value:

```
1.26
```

The following example:

```
RetVal = RoundValue(1.2568, 0)
```

returns the value:

```
1
```

RTrim()

Internal Basic syntax

Function RTrim(strString As String) As String

Description

Removes all trailing spaces in a string.

Input parameters

- **strString**: String to process.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
' 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()

Internal Basic syntax

Function Second(tmTime As Date) As Long

Description

Returns the number of seconds contained in the time expressed by the **tmTime** parameter.

Input parameters

- **tmTime**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strSecond
strSecond=Second(Date())
RetVal=strSecond :'Returns the number of seconds elapsed in the current ho
ur, for example "30" if the time is 15:45:30
```

SetMaxInst()

Internal Basic syntax

Function SetMaxInst(lMaxInst As Long) As Long

Description

This function enables you to set the maximum number of instructions that a Basic script can execute. By default, the number of instructions is limited to 10000.

Input parameters

- **lMaxInst**: This parameter contains the maximum number of instructions that can be executed by a script.

Output parameters

- 0: Normal execution.
- Other than zero: Error code.

Notes

 Note:

If you set the **lMaxInst** parameter to "0", the number of instructions that a script can execute is unlimited.

SetSubList()

Internal Basic syntax

Function SetSubList(strValues As String, strRows As String, strRowFormat As String) As String

Description

Defines the values of a sublist for a "ListBox" control.

Input parameters

- **strValues**: Source string containing the values of a "ListBox" control to be processed.
- **strRows**: List of values to add to or replace the characters contained in the string in the **strValues** parameter. The values are separated by the "|" character. The lines that are processed are identified by their identifier, situated to the right of the "=" sign. Unknown lines are not processed.

- **strRowFormat**: Formatting instructions for the sublist. Instructions are separated by the "|" character. This parameter has the following characteristics:
  - "1" represents the information contained in the first column of the sublist.
  - "i-j" can be used to define a group of columns.
  - "-" takes all columns into account.
  - An unknown column does not return a value.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
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") :'Returns "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")
:'Returns "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") :'Returns "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",
"-") :'Returns "a1|a2|a3=a0,B1|B2|B3|B4=b0,c1|c2|c3=c0"
MyStr=SetSubList("A|B|C,D|E|F", "X=*", "2") :'Returns "A|X|C,D|X|F"
RetVal=""
```

Sgn()

Internal Basic syntax

Function Sgn(dValue As Double) As Double

Description

Returns a value indicating the sign of a number.

Input parameters

- **dValue**: Number whose sign you want know.

Output parameters

The function can return one of the following values:

- 1: The number is greater than zero.
- 0: The number is equal to zero
- -1: The number is less than zero.

Example

```
Dim dNumber as Double
dNumber=-256
RetVal=Sgn(dNumber)
```

Shell()

Internal Basic syntax

Function Shell(strExec As String, bShowWindow As Long, bBackground As Long) As Long

Description

Launches an executable program.

Input parameters

- **strExec**: Full path of the executable to be launched.
- **bShowWindow**: If this parameter is set to 1 (default value), the command box is displayed when the program is launched. If this parameter is set to 0, the command box is not displayed.
- **bBackground** : If this parameter is set to 1 (default value), the function waits for the end of execution of the program before giving you back control (synchronous execution). If this parameter is set to 0, the program is executed asynchronously.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyId
MyId=Shell("C:\WinNT\explorer.exe")
RetVal=""
```

Sin()

Internal Basic syntax

Function Sin(dValue As Double) As Double

Description

Returns the sine of an number that is expressed in radians.

Input parameters

- **dValue**: Number whose sine you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Notes

Note:

The conversion formula for degrees to radians is the following:

```
angle in radians = (angle en degrees) * Pi / 180
```

Space()

Internal Basic syntax

Function Space(iCount As Long) As String

Description

Creates a string including the number of spaces indicated by the **iSpace** parameter.

Input parameters

- **iCount**: Number of spaces to be inserted into the string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyString
' Returns a string of 10 spaces.
MyString = Space(10)
' Inserts 10 spaces between two strings.
MyString = "Space" & Space(10) &  "inserted"
RetVal=MyString
```

Notes

Note:

This function can be used to format strings or to delete date in fixed length strings.

Sqr()

Internal Basic syntax

Function Sqr(dValue As Double) As Double

Description

Returns the square root of a number.

Input parameters

- **dValue**: Number whose square root you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim dCalc as Double
dCalc=Sqr(81)
RetVal=dCalc
```

Str()

Internal Basic syntax

Function Str(strValue As String) As String

Description

Converts a number to a string.

Input parameters

- **strValue**: Number to convert to a string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

StrComp()

Internal Basic syntax

Function StrComp(strString1 As String, strString2 As String, iOptionCompare As Long) As Long

Description

Compares two strings.

Input parameters

- **strString1**: First string.
- **strString2**: Second string.
- **iOptionCompare**: Comparison type. This parameter can be set to "0" for a binary comparison, or "1" for a text comparison.

Output parameters

- -1: **strString1** is greater than **strString2**.
- 0: **strString1** is equal to **strString2**.
- 1: **strString1** is less than **strString2**.

String()

Internal Basic syntax

Function String(iCount As Long, strString As String) As String

Description

String returns a string consisting of the **strString** character repeated over and over **iCount** times.

Input parameters

- **iCount**: Number of occurrences of the character **strString**.
- **strString**: Character used to compose the string.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim iCount as Integer
Dim strTest as String
strTest="T"
iCount=5
RetVal=String(iCount,strTest)
```

SubList()

Internal Basic syntax

Function SubList(strValues As String, strRows As String, strRowFormat As String) As String

Description

Returns a sublist of a list of values contained in a string representing the values of a "ListBox" control.

Input parameters

- **strValues**: Source string containing the values of a "ListBox" control to be processed.
- **strRows**: Identifiers of lines to be included in the sublist. The identifiers are separated by commas. Certain wildcard characters can be used:
  - "*" includes all identifiers in the sublist.
  - An unknown identifier returns an empty value for the sublist.
- **strRowFormat**: Formatting instructions for the sublist. Instructions are separated by the "|" character. This parameter has the following characteristics:
  - "1" represents the information contained in the first column of the list from which we are extracting a sublist.
  - "0" represents the identifier of the line in the list from which we are extracting a sublist.
  - "*" represents the information contained in all the columns (except the line identifier).
  - An unknown column does not return a value.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "a0,b0,a0", "3|2|3")
:'Returns "a3|a2|a3,b3|b2|b3,a3|a2|a3"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "*|0") :'Returns
"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") :'Returns
"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") :'Retur
ns "=a0,=b0,=c0"
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "z0", "*=0") :'Return
s ""
MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "*", "=1") :'Returns
"=a1,=b1,=c1"
```

```
MyStr=SubList("A|B|C,D|E|F", "*", "2=0") :'Returns "B,E"
RetVal=""
```

Tan()

Internal Basic syntax

Function Tan(dValue As Double) As Double

Description

Returns the tangent of a number expressed in radians.

Input parameters

- **dValue**: Number whose tangent you want to know.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

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

Notes

Note:

The conversion formula for degrees to radians is the following:

```
angle in radians = (angle en degrees) * Pi / 180
```

Time()

Internal Basic syntax

Function Time() As Date

Description

Returns the current time.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
RetVal = Time()
```

Timer()

Internal Basic syntax

Function Timer() As Double

Description

Returns the number of seconds elapsed since 12:00 AM.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
RetVal = Timer()
```

TimeSerial()

Internal Basic syntax

Function TimeSerial(iHour As Long, iMinute As Long, iSecond As Long) As Date

Description

This function returns a time formatted according to the **iHour**, **iMinute** and **iSecond parameters**.

Input parameters

- **iHour**: Hour.
- **iMinute**: Minutes.
- **iSecond**: Seconds.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

Each of these parameters can be set to a numeric expression representing a number of hours, minutes or seconds. Thus in the following example:

```
TimeSerial(12-8, -10, 0)
```

Returns the value:

```
3:50:00
```

When the value of a parameter is out of the expected range (i.e. 0-59 for minutes and seconds and 0-23 for hours), it is converted to the parameter the next up. Thus, if you enter "75" for the **iMinute** parameter, it will be interpreted as 1 hour and 15 minutes.

The following example:

```
TimeSerial (16, 50, 45)
```

Returns the value:

```
16:50:45
```

TimeValue()

Internal Basic syntax

Function TimeValue(tmTime As Date) As Date

Description

This function returns the time portion of a "Date+Time" value.

Input parameters

- **tmTime**: "Date+Time" format date.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

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

Returns the value:

```
15:00:00
```

ToSmart()

Internal Basic syntax

Function ToSmart(strString As String) As String

Description

This function reformats a source string by capitalizing the first letter of each word.

Input parameters

- **strString**: Source string to reformat.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The following example:

```
RetVal = ToSmart ("hello world")
```

returns the value:

```
Hello World
```

Trim()

Internal Basic syntax

Function Trim(strString As String) As String

Description

Returns a copy of a string with the leading and trailing spaces removed.

Input parameters

- **strString**: String to process.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
' 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()

Internal Basic syntax

Function UCase(strString As String) As String

Description

Returns a copy of a sting in which all lowercase characters are converted to uppercase.

Input parameters

- **strString**: Character string to convert to uppercase.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
' 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()

Internal Basic syntax

Function UnEscapeSeparators(strSource As String, strEscChar As String) As String

Description

Deletes all the escape characters from a string.

Input parameters

- **strSource**: Character string to process.
- **strEscChar**: Escape character to be deleted.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=UnEscapeSeparators("you\|me\|you\|", "\") :'Returns "you|me|you|"
RetVal=""
```

Union()

Internal Basic syntax

Function Union(strListOne As String, strListTwo As String, strSeparator As String, strEscChar As String) As String

Description

Merges two strings delimited by separators. Duplicates are deleted.

Input parameters

- **strListOne**: First string.
- **strListTwo**: Second string.
- **strSeparator**: Separator used to delimit the elements contained in the strings.
- **strEscChar**: Escape character. If this character prefixes the separator, it will be ignored.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim MyStr
MyStr=Union("a1|a2,b1|b2", "a1|a3,b1|b2", ",", "\") :'Returns "a1|a2,b1|b2
,a1|a3"
MyStr=Union("a1|a2,b1|b2", "a1|a3\,b1|b2", ",", "\") :'Returns "a1|a2,b1|b
2,a1|a3\,b1|b2"
RetVal=""
```

UTCToLocalDate()

Internal Basic syntax

Function UTCToLocalDate(tmUTC As Date) As Date

Description

This function converts a date in UTC format (time-zone independent) to a "Date+Time" format date.

Input parameters

- **tmUTC**: Date in UTC format.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
RetVal = UTCToLocaldate([DateTime])
```

Val()

Internal Basic syntax

Function Val(strString As String) As Double

Description

Converts a string representing a number to a double.

Input parameters

- **strString**: Character string to convert.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strYear
Dim dYear as Double
```

```
strYear=Year(Date())
dYear=Val(strYear)
RetVal=dYear :'Returns the current year
```

WeekDay()

Internal Basic syntax

Function WeekDay(tmDate As Date) As Long

Description

Returns the day of the week contained in the date expressed by the **tmDate** parameter.

Input parameters

- **tmDate**: Parameter in Date+Time format to be processed.

Output parameters

The number returned corresponds to a day of the week where "1" represents Sunday, "2" Tuesday, ..., "7" Saturday.

Example

```
Dim strWeekDay
strWeekDay=WeekDay(Date())
RetVal=strWeekDay :'Returns the day of the week
```

XmlAttribute()

Internal Basic syntax

Function XmlAttribute(strName As String, strValue As String) As String

Description

This function generates the strName="strValue" string, where strName remains unchanged and the predefined XML entities in strValue are converted into XML.

The five predefined XML entities and their conversions are as follows:

- The < entity corresponding to the < character
- The > entity corresponding to the > character
- The & entity corresponding to the & character
- The ' entity corresponding to the ' character
- The " entity corresponding to the " character

Input parameters

- **strName**: This parameter contains the name of the XML attribute.
- **strValue**: This parameter contains the value of the XML attribute.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

The example below:

```
RetVal = XmlAttribute("Equation & condition","ten < eleven")
```

Returns the value:

```
Equation & condition = "ten &lt; eleven"
```

Year()

Internal Basic syntax

Function Year(tmDate As Date) As Long

Description

Returns the year contained in the value expressed by the **tmDate** parameter.

Input parameters

- **tmDate**: Parameter in Date+Time format to be processed.

Output parameters

In case of error, a message is written to the Connect-It log.

Example

```
Dim strYear
strYear=Year(Date())
RetVal=strYear :'Returns the current year
```

III Index

Available functions - Full list of functions

- **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

Available functions - 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**

Available functions - 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