Asset Manager

Software Version: 9.62

Windows® and Linux® operating systems

Programmer reference

Document Release Date: December 2016

Software Release Date: December 2016

Hewlett Packard Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise 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. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise 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.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HP Passport and to sign in. To register for an HP Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com/.

Most of the support areas require that you register as an HP Passport user and to sign in. Many also require a support contract. To register for an HP Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HP Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPSW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is https://softwaresupport.hpe.com/km/KM01702731.

Contents

Part I: Introduction

Chapter 1: Programming fundamentals

This chapter presents the programming fundamentals using the Basic language that is available in Asset Manager. If you already have experience in programming and have used other languages, most of the information presented in this chapter will be familiar to you. However, we recommend reading through this chapter as certain common functions have been voluntarily omitted or limited in Asset Manager's Basic implementation.

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 explicit declaration of variables in Asset Manager 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	Meaning
Integer	Integer from -32 768 to +32 767.
Long	Integer from -2 147 483 648 to +2 147 483 647.
Single	4-bype floating-point number (single precision).
Double	8-byte floating-point number (double precision).
String	Text in which all characters are accepted.
Date	Date or Date+Time.
Variant	Generic type that can represent any other type.

> **Note:** These types are not available from an external tool. Only the types **Long**, **Double** and **String** are available. **Variant** does not exist and **Integer** and **Date** type objects are represented by a **Long**.

Numerical types

The Basic language available in Asset Manager 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** (2 147 483 647 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 Asset Manager 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, but any expression giving a numerical result can be used. This value will then be interpreted as **True** or **False** by Basic. **False** corresponds to the numerical value 0, all other values are considered as **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 <List of values 1>
<Instructions1>]
[Case <List of values 2>
<Instructions2>]
...
[Case Else
<Instructionsn>]
End Select
```

Each list of values contains a list of values separated by commas. If several **Case** keywords have values matching the test results, only the instruction set associated with the first matching **Case** will be executed.

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

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 named 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>]
```

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

> **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, and so on) on variables or compare them. There are several different types of counters:

- Assignment operators,
- Arithmetic operators,

- Relational operators (also named assignment operators),
- Logical operators.

Assignment operators

This type of operator enables you to assign a value to a variable. Asset Manager Basic uses one single assignment operator, the "=" sign. The assignment 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>`
<>	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

Asset Manager Basic enables simplified file management. The most common operations (read, write, and so on) are available as standard.

Reminder concerning files

A file is a way in which a program sees an 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, and so on). A logical record represents the minimum set of data that can be manipulated by a single elementary operation.

Asset Manager can only handle so-named 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 number>
```

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 that 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 open in read-write mode. This access type is only available for the**Append** access mode.
<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:

- Files must be opened using the **Open** clause before any read or write operations on the file.
- In **Append** or **Output** mode, if the referenced file does not exist, it is created.
- In **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**

The Input clause

This clause is used to read a given number of characters from a file open in **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>
```

> **Note:** 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>]
```

Chapter 2: Classification of functions

Functions are classified according to three different levels. A given function can be classified by:

Families of functions

Functions in the Asset Manager environment can be organized into several main families:

- Functions recognized by Asset Manager These are essentially functions that can be used in the scriptable parts (in Basic) of the software.
- Functions recognized by the Asset Manager API: These functions can be called by external tools or be a program written in a high-level language.

These main families of functions are not mutually exclusive. For example, certain Asset Manager API functions can be used in the Basic scripts in the software. Such a function, originating from the Asset Manager API is said to be "exposed" in Asset Manager's internal Basic scripts. The syntax of such a function may change but its behavior remains the same.

Scope of application of functions

The functions described in this document can be used in at least one of the following contexts:

- Asset Manager API libraries. In particular, the functions are available for development of Get-It applications.
- Field or link configuration script (**Configure the object** popup menu item or Asset Manager Application Designer) and by extension **Calculation script** (SQL name: memScript) of a calculated field:

  - Default value,
  - Mandatory nature,
  - Historization,
  - Read-only nature,
  - ...
- Script type action:
  - Script defined in the **Script of the action** (SQL name: Script) of a Script action.
- Asset Manager wizards:
  - "FINISH.DO" script of a wizard.
  - Value definition scripts for the properties of nodes.

Application modules

Each function is associated with one or more application modules. An application module describes the nature of operations carried out by the function. The different application modules are listed below:

- Built-in: Classic Basic functions, conversion and string handling functions, and so on.
- Technical: Connection to a database, handling of table, field, link, index, record and query objects.
- Functional: Generic, line of business functions.
- Cable.
- Procurement.
- Helpdesk.
- Chargeback.
- Wizards.
- Actions.
- Graphics.

Chapter 3: Conventions

This chapter describes:

Notation

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

Convention	Description
[]	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, they must appear in the script as shown below: **[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 script block spanning several lines 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:

Convention	Description
Fixed width characters	DOS command, function parameter are 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.

Format of Date and 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.

About dates

Dates are expressed differently in internal Basic and from external tools:

- 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).
- Externally, dates are expressed as a long integer ("Long" type) that represents the number of seconds elapsed since 01/01/1970 at midnight, independent of time zones (UTC time).

Format of Duration type constants in scripts

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

```
RetVal=259200
```

Likewise, functions that calculate durations, such as the "AmWorkTimeSpanBetween()" function, return a number of seconds.

> **Note:** In financial calculations, Asset Manager 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).

Chapter 4: Definitions

This chapter groups together the definitions of several essential terms.

You will find the following definitions:

Definition of a function

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

Here is an example of the syntax used to call an internal Asset Manager function:

AmConvertCurrency(strSrcName As String, strDstName As String, dVal As Double) As Double

Here is the syntax of the same function via the Asset Manager API:

double AmConvertCurrency(long hApiCnxBase,long ltm, const char *pszSrcName, const char *pszDstName,double dVal)

Definition of the CurrentUser virtual link

Definition

CurrentUser can be considered as a link starting in all tables and pointing to the record in the table of departments and employees corresponding to the current user.

- In the **CurrentUser** format, it points to the record corresponding to the current user, and returns the description string from the Employees and Departments table.
- In the **CurrentUser.Field** format, it returns the value of the field for the current user.

> **Note:** This virtual link is not displayed in the list of fields and links; therefore it is not directly accessible in Asset Manager's internal script builder. You must enter this expression manually.

Equivalencies

The **AmLoginName()** and **AmLoginId()** functions, which return the current user's **Name (SQL name: Name)** and ID (SQL name: lPersId), respectively, may be considered as functions derived from **CurrentUser**. In effect, the following are equivalent:

- AmLoginName()=[CurrentUser.UserLogin]
- AmLoginId()=[CurrentUser.lEmplDeptId]

Restrictions

CurrentUser will only work if a context is defined (the context being a table).

If there is no context, you must use another function.

Example:

You want to create a non-contextual action that executes a file whose path depends on the user connected to the Asset Manager database.

If the action were contextual, you would be able to create an **Executable** type action with the **Folder** field set to, for example: **c:\scripts\[CurrentUser.Name]**.

However, when an **Executable** type action does not have a context, **[CurrentUser.Name]** is considered to be fixed text.

You must therefore find another solution such as creating a **Script** type non-contextual action using the script:

```
RetVal = amActionExec("program.exe","c:\scripts\" + amLoginName())
```

Definition of a handle

A handle represents a unique identifier for an object. In the context of Asset Manager, this object can be a field, link, index, query, record, table or a connection. Handles are 32-bit integers ("Long" type).

> **Note:** The NULL value is not a valid handle.
>
> From external tools, you can also access (database) connection handles.

Definition of an error code

When a function fails, it returns an error code.

From external tools

This error code and associated message can be recovered by external tools via the "AmLastError()" and "AmLastErrorMsg()" functions respectively. It can be cleared using the "AmClearLastError()" function.

> **Note:** Any new function call clears the error code and previous message.

Internally

Internally (in Basic scripts, for example), the last error code and its description can be recovered using the **Err.Number** and **Err.Description** functions.

> **Note:** Internally, you don't need to program your own error handling. A script with problems will stop and a database rollback will be performed if necessary.

You can raise an error on purpose using the Err.Raise function. Its syntax is as follows:

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

> **Note:** When the creation or modification of a record is invalidated by the value of the "Validity"

> field for the table in question, it is a good idea to raise an error message using the **Err.Raise** function in order to warn the user (code 12006 or 12007). If you do not do this, the user will not necessarily understand why the record cannot be modified or created.

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.
12005	Internal database server error
12006	Invalid value (incorrect type for a parameter, and so on)
12007	Non valid record (a mandatory field is not populated, for example)
12008	Problems with database access rights
12009	Obsolete or non implemented function
12010	Maximum number of database connections exceeded

Chapter 5: Function and parameter types

This chapter contains information on the following:

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 -2,147,483,648 to +2,147,483,647.
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.

> **Note:** Not all of these types are available from external tools. Only Long, Double and String types are available. Variant is not used and Integer and Date objects are represented by a Long.

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 the cause of compilation and runtime errors in your programs.

For example, you cannot use a function returning a certain typed value in the definition of the default value of a differently typed field. Try, for example, assigning this default value script to any "Date" or "Date and time" type field:

```
RetVal=AmLoginName()
```

The "AmLoginName()" function returns the name of the connected user in the form of a character string ("String" type). The type of this return value is therefore incompatible with "Date" type fields and Asset Manager displays an error message.

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. To avoid any possible confusion, the prefixes used in this reference differ according to the syntax (API or Basic) of the function. The following table resumes the equivalencies between the prefixes used in the API syntax and the Basic syntax:

Type	Prefix used in the API syntax	Prefix used in the Basic syntax
Integer	"i"	"i"
Long	"h" for a handle or "l" for a number	"l"
Double	"d"	"d"
String	"char*psz"	"str"
Date	"ltm"	"dt"
Variant	"v"	"v"

Part II: Using the API

Chapter 6: Introduction

The Asset Manager API is provided as a DLL, useable in Microsoft Windows operating systems.

The following environments are supported:

- Microsoft Visual Studio 2008 Professional Edition
- Microsoft Visual Studio 2008 Standard Edition
- Microsoft Visual Studio Team System 2008
- Microsoft Visual Studio 2010 Premium
- Microsoft Visual Studio 2010 Professional
- Microsoft Visual Studio 2010 Ultimate
- Microsoft Visual Studio Premium 2012
- Microsoft Visual Studio Professional 2012
- Microsoft Visual Studio Ultimate 2012
- Microsoft Visual Studio Premium 2013
- Microsoft Visual Studio Professional 2013
- Microsoft Visual Studio Ultimate 2013
- Visual Studio Professional 2015
- Visual Studio Enterprise 2015

Caution: The entry points in the library (.dll) are not provided for .NET environments. If you wish to use these development environments, you must define these entry points yourself.

Note: The API should be compatible with all tools authorizing the use of third-party DLLs.

Note: Before you use the Asset Manager API, we strongly recommend that you add the Asset Manager binary directory to the system variables. For example:

- For a 32-bit operating system, add `<Asset Manager Install Dir>/bin` to the system variables.
- For a 64-bit operating system, add `<Asset Manager Install Dir>/x64` to the system variables.

Warning

Before using the Asset Manager API, the user should be familiar with the terminology used in the Asset Manager conceptual model. In particular, a minimal knowledge of the database structure is required.

Information on the structure of the database can be found in the manual entitled "Reference guide: Administration and advanced use", chapter "Structure of the database" and in the "Database.txt" and "Tables.txt" files, which can be found in the "doc\infos" sub-folder of the Asset Manager installation folder.

Installation

Before using the Asset Manager API, it is highly recommended to install a fully functional version of Asset Manager. In this way, you can quickly test whether databases can be correctly accessed from a given computer and create or configure database connections. The API uses the same database layers and the same configuration information as Asset Manager to access data sources, so problems can often be investigated from within Asset Manager.

The typical steps for setting up a development environment with Asset Manager are as follows:

- Install a 32-bit version of Asset Manager with the Asset Manager API package.
- Use Asset Manager to configure the data source, and try to open a database.
- Use your development environment to call Asset Manager API functions.

To familiarize yourself with the Asset Manager API, we recommend using a demonstration database or any non critical source of data for which manipulation errors are not critical.

.ini configuration file associated with the DLL

For information, see the **.ini and .cfg files** chapter in the *HP Asset Manager Installation and upgrade* document.

Refer to the following sections in particular:

- **Available .ini and .cfg files**
- **Controlling the modification of the .ini files**

Chapter 7: Methodology

A typical sequence of operations using the Asset Manager API would be:

1. Create a query using an AQL statement:

   ```
   SELECT AssetTag, User.Name, Supervisor.Name FROM amPortfolio
   ```

   > **Note:** You can also use Asset Manager Export Tool to generate an AQL query.

2. Browse the query result set and retrieve any useful handles on specific items.
3. Use the retrieved handles to update the information in the corresponding objects.
4. Commit (accept) or rollback (cancel) the whole transaction.

Chapter 8: Concepts and examples

This section contains information on the following:

Concepts

Asset Manager is built around an object oriented design and the API maintains this structural view. To accommodate the limitation of Windows DLLs which imposes the use of a flat "C like API", the Asset Manager API work around this problem by using handles (32-bit integers) to identify every user-created object. This approach has the advantage of allowing non-object oriented languages to access the Asset Manager object model.

Before doing anything else, your program must call "AmStartUp()" in order to initialize the Asset Manager libraries. Your program must also terminate by calling the "AmCleanUp()" function.

Before accessing a database object, a connection should be established between the user and the database. This connection is identified by a "handle" on a "connection" object (this handle is then used in all the API functions that interact with the database. It corresponds to the parameter "hApiCnxBase". This object can then be used to create queries and gain access to records.

> **Note:** All database objects are linked to a connection so information about user privileges can be checked.

The first step is to open a connection using a valid data source name and a valid login/password combination.

> **Caution:** When you connect to the Asset Manager database via the Asset Manager API, a connection slot is used.

Handling dates

When reading dates, you have the choice of the following two functions for "Date" and "Date and time" type fields:

- "AmGetFieldLongValue()" which returns the date as a Unix "Long" (UTC). We recommend using this function for calculations involving dates.
- "AmGetFieldStrValue()" which returns the date as a string in the same format as the Windows Control Panel. This date takes time zones into account. We recommend using this function when you need to display a date.

First example

The following example, written in C, declares a connection to the demonstration database:

```
handle lCnx ;
lCnx = AmGetConnection(AMDemo95en, Admin , ,Admin) ;
```

"lCnx" is a connection handle that can be used to identify the newly created connection.

This connection can now be used to create queries and access the database. The following example, written in C, defines a query on the table of assets and browses the results set:

```
#include apiproto.h
#define SZ_MODEL_LEN 200
handle lCnx ;
handle lQuery ;
handle lStatus ;       /* to store error code */
char szModel[SZ_MODEL_LEN] ;
/* dll initialization */
AmStartup();
/* Open a connection */lCnx = AmGetConnection("AMDemo95en","Admin" ,"", "Admin") ;
if( lCnx != 0 )
{
/* Creation of a query object */
lQuery = AmQueryCreate (lCnx);
if( lQuery != 0 )
{
/* Construction of  the result set : all assets from Compaq*/
lStatus = AmQueryExec(lQuery, "select AssetTag from amAsset where Model.brand.name
= 'Compaq'");
```

```
/* Navigates through the result set */
while( !lStatus )
{
/* Read the first field (AssetTag) of the current item in the query */
lStatus = AmGetFieldStrValue(lQuery,0,szModel,SZ_MODEL_LEN-1);
if( lStatus == 0 )
{
printf(" Compaq AssetTag=%s\n",szModel);
lStatus = AmQueryNext(lQuery);
}
}
/* clean things up */
AmReleaseHandle(lQuery);
}
AmReleaseConnection(lCnx);
}
AmCleanup();
```

Second example

Queries are used to locate objects in the database. When you want to update a record, a handle on a "record" object must be obtained using a query. The record can then be processed using other Asset Manager API functions.

The next example shows how to modify a field in a specific record:

```
/* Handles for objects */
handle lCnx ;
handle lQuery ;
handle lStatus ;
handle lRecord ;
AmStartup();
lCnx = AmGetConnection("AMDemo95en","Admin" ,"", "Admin") ;
/* Creation of a query object attached to lCnx */
lQuery = AmQueryCreate(lCnx);
/* Mark the starting point of the current transaction */
AmStartTransaction(lCnx);
/* Use a query that matches a single object */
lStatus = AmQueryGet(lQuery, "select model, AssetId where brand = 'Compaq' and
barcode='34234'") ;
/* Get a record handle to the matching object */
lRecord = AmGetRecordHandle(lQuery) ;
/* Change the field Field1 with new value spam */
lStatus = AmSetFieldStrValue(lRecord, "Field1", "Spam");
/* Update the change for the current session */
lStatus = AmUpdateRecord(lrecord);
```

```
/* Commit all modifications to the database */
lStatus = AmCommit(lCnx) ;
/* you can release here query and record objects */
/* but closing connection will do it */
/* Close the connection to the database */
AmReleaseConnection(lCnx);
AmCleanup();
```

This example shows how to get a unique record handle using the query mechanism. In this sample code, the query is used to locate a single item, but it is also possible to use "AmQueryExec()" to get a set of records and then get a record handle for one or more records.

> **Note:** For reasons of simplicity, this example does not deal with all possible error codes.

Part III: Web Services

Chapter 9: Web Services

This chapter includes:

Introduction to Asset Manager Web services 66
Checking the definition of the Web services 67
API naming conventions 68

Development with Flash using calls to the WSDL: Limitations 71
Calling the Asset Manager WSDL from HP Service Manager: Limitation 72
Re-initializing the connection pool 73

Introduction to Asset Manager Web services

Asset Manager can publish Web services.

To do this, Asset Manager uses the SOAP protocol.

Published Web services let you communicate easily with the Asset Manager server

This enables you to execute read actions (such as **retrieveAllPurchaseRequest**) and write actions (for example, **savePurchaseRequest**).

These actions can be performed via development environments such as Microsoft Studio 2003 ASP.Net, Java + Ant or any other tool that is capable of interacting with Web services.

> **Note:** Asset Manager cannot consume (call) third-party Web services.
>
> You can use HPE Connect-It to call third-party Web services.

Web services published by Asset Manager Web Service are grouped according to functional domain (only functional domains whose **WEB service** (seWebService) field equals **autonomous** are retained). They include functional sub-domains whose **seWebService** field is equal to **From within parent domain**.

The Web services publish the objects from the Asset Manager database (screens, actions, and so on).

The published Web services can contain a large number of APIs.

To access the definition of a given Web service, enter a URL similar to the following:

```
http://<Name of the Asset Manager Web Service server>:<Asset Manager Web Service
port>/AssetManagerWebService/services/Head/<Name of the Web service>?WSDL
```

<Name of the Web service> corresponds to the SQL name of a functional domain whose **WEB service** (seWebService) field equals **autonomous**.

For additional information on using Web services, see "Example code used to call Web services" on page 70.

Checking the definition of the Web services

Tagging is done in order to track the state of the objects manipulated by the Web services at a given moment in time. Using tagging, you can force applications that access the database, but do not do so via the Windows client, to use objects that correspond to the state of the database at a given moment in time.

To learn how to tag the Web services, read the **Tailoring** guide, chapter **Customizing the database**, **Development best practices/ Tag the Web services** section of the .**Tailoring** guide, chapter **Customizing the database**, section **Development best practices/ Tag the Web services**.

API naming conventions

> **Note:** APIs exposed by the Web services are organized around documents, not records.
>
> A **PurchaseRequest** document contains all of the associated request lines.

Below is the list of naming conventions for the APIs that are exposed by the Web services:

- **retrieveAllXxxListByYyy**

  Retrieves a list of **Xxx** type documents filtered by **Yyy**.

  Xxx is generated from the SQL name of the screens.

  Yyy is generated from:

  - The SQL name of the fields and links that make up an index (for example, **AssetAnddCntrIncluded**)
  - The query's SQL name
  - The screen set, for query wizards (QBE fields)

  For example: **retrieveAllPurchaseRequestListByUser**

- **retrieveFirstXxxListByYyy**

  Retrieves the list of the first n **Xxx** type documents filtered by **Yyy** (n is a parameter of the API).

- **retrieveNextXxxList**

  Retrieves the list of n **Xxx** type documents after the document passed as parameter.

- **retrievePreviousXxxList**

  Retrieves the list of n **Xxx** type documents before the document passed as parameter.

- **retrieveLastXxxListByYyy**

  Retrieves the list of the last n **Xxx** type documents filtered by **Yyy** (n is passed as parameter).

- **retrieveXxxByYyy**

  Retrieves 1 **Xxx** type document filtered by **Yyy**.

- **retrieveXxx**

  Retrieves 1 **Xxx** type document from an API reference passed as parameter.

- **retrievePageXXXXList**

  Retrieves the specified range of XXX type documents, the **BeginPos** and **EndPos** of the range can be passed as parameters.

- **saveXxx**

  Saves 1 **Xxx** type document.

- **deleteXxx**

  Deletes 1 **Xxx** type document.

- **countXxx**

  Counts the number of **Xxx** type documents corresponding to the list passed as parameter (is not limited to the size of the list loaded in memory).

- **retrieveXxxBreakdown**

  Generates a breakdown of **Xxx** type documents corresponding to the list passed as parameter (is not limited to the size of the list loaded in memory).

- **executeZzz**

  Executes the **Zzz** action.

The ListScreen parameter

In Asset Manager versions greater than 9.32, a parameter named **ListScreen** is added to the following Web Service APIs:

- **retrieveAllXxxListByYyy**
- **retrieveFirstXxxListByYyy**
- **retrieveNextXxxList**
- **retrievePreviousXxxList**
- **retrieveLastXxxListByYyy**

For example, in the C# code, the definition of the Web Service API **retrieveFirstLocationListByBarCode** is changed from

```
public LocationList retrieveFirstLocationListByBarCode(int MaxNumberOfRecord,
string BarCode, bool UseBarCodeAsPattern, LocationListBase OriginalList,
LocationOrder Order)
```

to

```
public LocationList retrieveFirstLocationListByBarCode(int MaxNumberOfRecord,
string BarCode, bool UseBarCodeAsPattern, LocationListBase OriginalList,
LocationOrder Order, bool ListScreen)
```

The type of new parameter **ListScreen** is Boolean:

- If it is set to true: Retrieve data with optimized queries.
- If it is set to false: Retrieve data with non-optimized queries.

If you have a program developed based on the old Asset Manager Web Service APIs, compiling the program with the above APIs will fail. You must update the code to accommodate the changes to the Web Service APIs.

Example code used to call Web services

Version 9.62 is supplied with sample projects whose code calls the Asset Manager Web services.

These projects are located in the **samples\ws** folder of the Asset Manager installation folder.

These projects have been designed in the following environments:

Microsoft ASP.Net

- **RequestSample**

  C# ASP.Net project used to display the list of purchase requests and to create a request.

- **ChartingSample** This VB.Net WindowsForms project is used to display a graph showing the breakdown of expense lines by cost type.

  > **Note:** This code requires you install the **DotNetCharting** component that can be downloaded from:
  >
  > http://www.dotnetcharting.com/download.aspx

- **ACPhoneListSample**

  This C# WindowsForms project is used to display the Asset Manager database directory using paging functions (the records are returned by group instead of all at the same time).

- **UpdatePortfolioLocation**

This project is a sample for using Asset Manager Web Service in C# to update the location of a portfolio.

Java + Ant

- **RSS**

  This project is used to display news and workflow tasks that have been assigned to the connected user via RSS feeds (Really Simple Syndication).

  > **Note:** The RSS format is a way to describe the contents of a Web site (articles, information, events) and, in general, any page that provides content which is updated on a regular basis.
  >
  > It allows Web sites to automatically display the latest information published on another site.
  >
  > The RSS format is used to share content between Web sites.
  >
  > RSS content can be read by aggregators, which are specialized RSS feed readers.

- **CoreServiceSample**

  This project is used to display the list of employees and departments from the demo database in a DOS console.

Development with Flash using calls to the WSDL: Limitations

> **Note:** The workaround described in this section has been tested with Flash 8.

If you use Flash 8 applications that call the Asset Manager WSDL, you must implement a workaround.

Do the following:

1. Start Internet Explorer.
2. Access the Asset Manager Web Service page (http://<Asset Manager Web Service server

name>:<Asset Manager Web Service port>/AssetManagerWebService).

3. Display the desired revision of the Web services for which you want to implement the Flash application.

   For example: **R50**.

4. For each domain (for example: **Administration**):

   a. Click the **schema** link.Save the document that is displayed to a local folder (for example: **C:\FlashDev\schema\R50\Administration\Administration.wsdl**).

   b. Click the **wsdl** link.Save the document that is displayed to a local folder (for example: **C:\FlashDev\schema\R50\Administration\AdministrationTypes.xsd**).

   c. Open each of the **.wsdl** and **.xsd** files.

      Modify the lines that start with **schemaLocation=** (for example: **schemaLocation="../../schema/R50/Administration/AdministrationTypes.xsd"**).

      Replace the relative path **../../** with the absolute path.

      For example:

      ```
      schemaLocation="file:///C|/FlashDev/schema/R50/Administration/
      AdministrationTypes.xsd"
      ```

   d. Develop the Flash object using the local WSDL.

Calling the Asset Manager WSDL from HP Service Manager: Limitation

The Asset Manager WSDL contains definitions that are not supported by HP Service Manager.

This prevents HP Service Manager from calling them.

Re-initializing the connection pool

When one of the following operations is performed:

- Add, modify or delete a record in the **Itemized list values** (amItemListVal) or **Itemized lists** (amItemizedList) tables via the Windows or Web client.
- Add a column to the **ColName** property of a wizard's **DBLISTBOX** control and the column is not part of the default columns used for the source table's screens (Asset Manager Application Designer/ Detail of the source table/ Detail of the screens/ **List/Detail** tab/ **Columns of the list** and **Other columns** fields).

... you must re-initialize the connection pool using Asset Manager Web Service in order for these operations to be taken into account by the Web clients:

1. Start Asset Manager Web Service:

   ```
   http://<Name or IP address of the Asset Manager Web Service
   server>:<Asset Manager Web Service port>/AssetManagerWebService
   ```

2. Click this link: Reset the connection pool.

   You must have administration rights to perform this operation.

   > **Tip:** You must do this because these records are stored in a cache that needs to be refreshed.

Part IV: Alphabetical Reference

Abs()

Returns the absolute value of a number.

Internal Basic syntax

```
Function Abs(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

AmActionDde()

This function sends a DDE request to a DDE server application. Using this function, Asset Manager can control another application using a DDE link. This function is equivalent to a DDE type action.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

API syntax

```
long AmActionDde(handle hApiCnxBase, char *strService, char *strTopic,
char *strCommand, char *strFileName, char *strDirectory, char
*strParameters, char *strTable, long lRecordId);
```

Internal Basic syntax

```
Function AmActionDde(strService As String, strTopic As String, strCommand
As String, strFileName As String, strDirectory As String, strParameters
As String, strTable As String, lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strService*: This parameter contains the name of the DDE service provided by the executable you want to call. Refer to the documentation of the executable to obtain the list of DDE services it provides.
- *strTopic*: This parameter contains the topic (that is, the context) in which a DDE action must be executed.
- *strCommand*: This parameter contains the commands the external application must execute. You must follow the syntax defined by the external application.
- *strFileName*: If the service is not resident in memory, you must load it by specifying in this parameter the name of the executable (or the name of any file associated with an executable using the Windows File Manager) which activates the service.
- *strDirectory*: This parameter contains the path for the file defined in *strFileName*.
- *strParameters*: This parameter contains the various parameters to pass to the executable which activates the service when it is launched.
- *strTable*: Optional parameter used when the action is contextual. It indicates the SQL name of the table containing the record to which the action applies.
- *lRecordId*: Optional parameter used when the action is contextual. It indicates the identifier of the record to which the action applies.

Output parameters

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

AmActionExec()

This function launches an ".exe", ".com", ".bat", ".pif" application. You can also refer to any type of document, as long as the document extension is associated with an executable via the Windows File Manager. This function is equivalent to an action of "Executable" type.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

API syntax

```
long AmActionExec(handle hApiCnxBase, char *strFileName, char
*strDirectory, char *strParameters, char *strTable, long lRecordId);
```

Internal Basic syntax

```
Function AmActionExec(strFileName As String, strDirectory As String,
strParameters As String, strTable As String, lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strFileName*: The following list summarizes the possible behaviors of the function for the Windows and Web clients depending on the situation:
  - http or https:
    - Function triggered by a Windows client: Starts the default Internet browser on the client's Windows workstation and connects to the URL address provided by the *strFileName* parameter.
    - Function triggered by a Web client: Displays the page corresponding to the URL address specified by the *strFileName* parameter in the Web client's workspace.
  - ftp:
    - Function triggered by a Windows client: Starts the Windows explorer on the client's Windows workstation and connects to the ftp site provided by the *strFileName* parameter.

        - Function triggered by a Web client: Displays the ftp site corresponding to the URL address specified by the *strFileName* parameter in the Web client's workspace.
    - mailto:
        - Function triggered by a Windows client: Starts the default messaging application on the client's Windows workstation and creates a new message using the parameters specified after **mailto:** in the *strFileName* parameter
        - Function triggered by a Web client: Starts the default messaging application on the Web client's workstation and creates a new message using the parameters specified after **mailto:** in the *strFileName* parameter
    - Other values:
        - Function triggered by a Windows client: Executes the file specified by the *strFileName* parameter.
        - Function triggered by a Web client: The action's behavior depends on the value of the **Actions/ Executing executable-type actions** (ExecuteAction) database option (**Administration/ Database options...** menu of the Windows client):
            - None: The file is not executed on the Asset Manager Web Service station and an error is returned.
            - Server: The file is executed on the Asset Manager Web Service station.
            - Client: A message is displayed by the Web client describing the action that could have executed on the Web client.
- *strDirectory*: This parameter contains the path for the file specified in the *strFileName* parameter.
- *strParameters*: This optional parameter contains the various parameters to be provided to the executable when it is launched.
- *strTable*: Optional parameter used when the action is contextual. It indicates the SQL name of the table containing the record to which the action applies.
- *lRecordId*: Optional parameter used when the action is contextual. It indicates the identifier of the record to which the action applies.

Output parameters

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

Example

This example executes the Windows NT explorer (situated in the "WinNT" folder of the "C" drive):

```
RetVal=AmActionExec("explorer.exe", "c:\winnt\" )
```

AmActionMail()

This function sends a message via one of the messaging systems managed by Asset Manager:

- Internal messaging system.
- External messaging system based on the VIM standard (Lotus Notes, and so on)
- External messaging system based on the MAPI standard (Microsoft Exchange, Microsoft Outlook, and so on)
- External messaging system based on the SMTP standard (Internet standard)

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

API syntax

```
long AmActionMail(handle hApiCnxBase, char *strTo, char *strCc, char
*strCcc, char *strSubject, char *strMessage, long iPriority, long
bAcknowledge, char *strRefObject, char *strTable, long lRecordId, long
bHtml);
```

Internal Basic syntax

```
Function AmActionMail(strTo As String, strCc As String, strCcc As String,
strSubject As String, strMessage As String, iPriority As Long,
bAcknowledge As Long, strRefObject As String, strTable As String,
lRecordId As Long, bHtml As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTo*: This parameter contains the list of addresses for the message recipients in the form messaging_system:address. The semi-colon is used as a separator.
- *strCc*: This parameter contains the list of addresses for people who are copied in the message. The semi-colon is used as a separator.
- *strCcc*: This parameter contains the list of addresses for people who receive a blind copy of the message (they do not appear in the list of recipients). The semi-colon is used as a separator.
- *strSubject*: This parameter contains the message subject.
- *strMessage*: This parameter contains the message body.
- *iPriority*: This parameter defines the priority for sending the message:
  - 0: Low priority
  - 1: Normal priority.
  - 2: High priority.
- *bAcknowledge*: This parameter indicates whether the message sender will receive an acknowledgement:
  - 0: the sender does not receive an acknowledgement.
  - 1: the sender does receive an acknowledgement.
- *strRefObject*: This parameter is only used for messages sent via the Asset Manager internal messaging system. It contains the SQL name of the link to follow from the record corresponding to the context of execution to reach the referenced object. The CurrentUser virtual link can be used.
- *strTable*: Optional parameter used when the action is contextual. It indicates the SQL name of the table containing the record to which the action applies.

- *lRecordId*: Optional parameter used when the action is contextual. It indicates the identifier of the record to which the action applies.

bHtml: This parameter indicates whether the message is sent in HTML format:

- 0: the message is sent in plain text.
- 1: the message is sent in HTML format.

Output parameters

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

AmActionPrint()

This function prints a report on a given record in the database.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

Internal Basic syntax

```
Function AmActionPrint(lReportId As Long, lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lReportId*: This parameter contains the identifier of the report to be printed.
- *lRecordId*: This parameter contains the identifier of the record concerned by the report. By default, this parameter is set to "0". The table concerned is implicitly defined by the report.

Output parameters

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

AmActionPrintPreview()

This function triggers a print preview of a report concerning a given database record.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

Internal Basic syntax

```
Function AmActionPrintPreview(lReportId As Long, lRecordId As Long) As
Long
```

Field of application

Version: 3.60

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lReportId*: This parameter contains the identifier of the report concerned.
- *lRecordId*: This parameter contains the identifier of the record concerned by the report. By default, this parameter is "0".

Output parameters

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

AmActionPrintTo()

This function prints a report on a given database record and on a given printer.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

Internal Basic syntax

```
Function AmActionPrintTo(strPrinterName As String, lReportId As Long,
lRecordId As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strPrinterName*: This parameter contains the name of the printer to use.
- *lReportId*: This parameter contains the identifier of the report to print.
- *lRecordId*: This parameter contains the identifier of the record concerned by the report. By default, this parameter is set to "0".

Output parameters

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

AmAddAllPOLinesToInv()

This function adds a purchase order in full to an existing supplier invoice.

API syntax

```
long AmAddAllPOLinesToInv(handle hApiCnxBase, long lPOrdId, long lInvId);
```

Internal Basic syntax

```
Function AmAddAllPOLinesToInv(lPOrdId As Long, lInvId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	

	Available
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdId*: This parameter contains the identifier of the order to be added to the supplier invoice.
- *lInvId*: This parameter contains the identifier of the supplier invoice to which the purchase order is added.

Output parameters

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

AmAddCatRefAndCompositionToPOrder()

This function enables you to add the full contents of a catalog reference to a given purchase order.

API syntax

```
long AmAddCatRefAndCompositionToPOrder(handle hApiCnxBase, long
lPOrderId, long lCatRefId, double dCatRefQty, long lRequestId, double
dUnitPrice, char *strCur);
```

Internal Basic syntax

```
Function AmAddCatRefAndCompositionToPOrder(lPOrderId As Long, lCatRefId
As Long, dCatRefQty As Double, lRequestId As Long, dUnitPrice As Double,
strCur As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrderId*: This parameter contains the identifier of the purchase order to complete.
- *lCatRefId*: This parameter contains the identifier of the catalog reference.
- *dCatRefQty*: This parameter contains the quantity (in the unit associated with the product) to add.
- *lRequestId*: This parameter contains the identifier of the request that this purchase order will satisfy.
- *dUnitPrice*: This parameter contains the unit price of the product of the catalog reference.
- *strCur*: This parameter contains the code of the currency in which the unit price is expressed

Output parameters

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

Notes

In particular, this function can enable you to use the composition of the products of a catalog reference to fill out a purchase order.

AmAddCatRefToPOrder()

This function enables you to add a catalog reference to an existing purchase order.

API syntax

```
long AmAddCatRefToPOrder(handle hApiCnxBase, long lRequestLineId, long lCatRefId, long lPOrderId, double dQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddCatRefToPOrder(lRequestLineId As Long, lCatRefId As Long, lPOrderId As Long, dQty As Double, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestLineId*: This parameter contains the identifier of the request line associated with the purchase order.
- *lCatRefId*: This parameter contains the identifier of the catalog reference to be added.
- *lPOrderId*: This parameter contains the identifier of the purchase order concerned by the operation.
- *dQty*: This parameter contains the quantity (in the unit associated with the product) to add.
- *bCanMerge*: This parameter enables you to specify whether the added line can be merged with an existing line in the purchase line.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAddEstimLinesToPO()

This function adds all estimate lines of an estimates to an existing order.

API syntax

```
long AmAddEstimLinesToPO(handle hApiCnxBase, long lEstimId, long lPOrdId,
long bMergeLines);
```

Internal Basic syntax

```
Function AmAddEstimLinesToPO(lEstimId As Long, lPOrdId As Long,
bMergeLines As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lEstimId*: This parameter contains the identifier of the estimate to be added to the purchase order.
- *lPOrdId*: This parameter contains the identifier of the purchase order to which the all the estimate lines of the estimate are added.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to give one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddEstimLineToPO()

This function adds an estimate line to an existing purchase order.

API syntax

```
long AmAddEstimLineToPO(handle hApiCnxBase, long lEstimLineId, long
lPOrdId, long bMergeLines);
```

Internal Basic syntax

```
Function AmAddEstimLineToPO(lEstimLineId As Long, lPOrdId As Long,
bMergeLines As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lEstimLineId*: This parameter contains the identifier of the estimate line to be added to the purchase order.
- *lPOrdId*: This parameter contains the identifier of the purchase order to which the estimate line is added.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddLicContentToRequest()

This function adds to a purchase request all software installations covered by a license.

API syntax

```
long AmAddLicContentToRequest(handle hApiCnxBase, long lRequestId, long
lLicModelId, long lParent, long bExternalParent);
```

Internal Basic syntax

```
Function AmAddLicContentToRequest(lRequestId As Long, lLicModelId As
```

```
Long, lParent As Long, bExternalParent As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestId*: This parameter contains the identifier of the purchase request concerned by the operation.
- *lLicModelId*: This parameter contains the identifier of the license model.
- *lParent*: This parameter contains the identifier of the portfolio item or request line that will be the parent of the request lines created.
- *bExternalParent*: If this parameter is set to "1", the parent of the lines created is an existing portfolio item in the request line.

Output parameters

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

Notes

In particular, this function can enable you to use the composition of the products of a catalog reference to fill out a purchase order.

AmAddPOLineToInv()

This function adds a given quantity of item(s) on an order line to a supplier invoice.

API syntax

```
long AmAddPOLineToInv(handle hApiCnxBase, long lPOrdLineId, long lInvId,
double dQty);
```

Internal Basic syntax

```
Function AmAddPOLineToInv(lPOrdLineId As Long, lInvId As Long, dQty As
Double) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdLineId*: This parameter contains the identifier of the order line to be added to the supplier invoice.
- *lInvId*: This parameter contains the identifier of the supplier invoice to which the items of the order line are added.
- *dQty*: This parameter contains the quantity of items on the order line to be added to the supplier invoice.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAddPOrderLineToReceipt()

This function enables you to add an order line to a receipt. In this way you can receive an order line within the existing receipt.

API syntax

```
long AmAddPOrderLineToReceipt(handle hApiCnxBase, long lPOrderLineId,
long lRecptId, double dQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddPOrderLineToReceipt(lPOrderLineId As Long, lRecptId As
Long, dQty As Double, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrderLineId*: This parameter contains the identifier of the order line.
- *lRecptId*: This parameter contains the identifier of the impacted receipt.
- *dQty*: This parameter contains the quantity to receive. In this way, you can limit the quantity received in relation to the quantity ordered (in the unit of the product).
- *bCanMerge*: This parameter enables you to specify whether the line can be merged with an existing line in the receipt.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAddReceiptLineToInvoice()

This function enables you to add a receipt line to an invoice. By doing so, you can invoice a receipt line within an existing invoice.

API syntax

```
long AmAddReceiptLineToInvoice(handle hApiCnxBase, long lRecptLineId,
long lInvoiceId, double dQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddReceiptLineToInvoice(lRecptLineId As Long, lInvoiceId As
Long, dQty As Double, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRecptLineId*: This parameter contains the identifier of the receipt line.
- *lInvoiceId*: This parameter contains the identifier of the impacted invoice.
- *dQty*: This parameter contains the quantity to invoice. In this way, you can limit the quantity invoiced in relation to the quantity received (in the unit of the product).
- *bCanMerge*: This parameter enables you to specify whether the line can be merged with an existing line in the invoice.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAddReqLinesToEstim()

This function adds all request lines of a request to an existing estimate.

API syntax

```
long AmAddReqLinesToEstim(handle hApiCnxBase, long lReqId, long lEstimId,
long bMergeLines);
```

Internal Basic syntax

```
Function AmAddReqLinesToEstim(lReqId As Long, lEstimId As Long,
bMergeLines As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the request to be added to the estimate.
- *lEstimId*: This parameter contains the identifier of the estimate to which all the request lines of the request are added.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddReqLinesToPO()

This function adds all the request lines of a request to an existing purchase order. The supplier specified in the request must be identical to that in the purchase order concerned.

API syntax

```
long AmAddReqLinesToPO(handle hApiCnxBase, long lReqId, long lPOrdId,
long bMergeLines);
```

Internal Basic syntax

```
Function AmAddReqLinesToPO(lReqId As Long, lPOrdId As Long, bMergeLines
As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lReqId*: This parameter contains the identifier of the request to be added to the purchase order.
- *lPOrdId*: This parameter contains the identifier of the purchase order to which the request lines are to be added.

- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddReqLineToEstim()

This function adds a request line to an existing estimate.

API syntax

```
long AmAddReqLineToEstim(handle hApiCnxBase, long lReqLineId, long
lEstimId, long bMergeLines);
```

Internal Basic syntax

```
Function AmAddReqLineToEstim(lReqLineId As Long, lEstimId As Long,
bMergeLines As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqLineId*: This parameter contains the identifier of the request line to be added to the estimate.
- *lEstimId*: This parameter contains the identifier of the estimate to which the request line is added.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddReqLineToPO()

This function adds a request line to an existing purchase order.

API syntax

```
long AmAddReqLineToPO(handle hApiCnxBase, long lReqLineId, long lPOrdId,
long bMergeLines);
```

Internal Basic syntax

```
Function AmAddReqLineToPO(lReqLineId As Long, lPOrdId As Long,
bMergeLines As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqLineId*: This parameter contains the identifier of the request line to be added to the purchase order.
- *lPOrdId*: This parameter contains the identifier of the purchase order to which the request line is to be added.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.

Output parameters

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

AmAddRequestLineToPOrder()

This function enables you to add a request line to a purchase order.

API syntax

```
long AmAddRequestLineToPOrder(handle hApiCnxBase, long lRequestLineId,
long lPOrderId, double dQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddRequestLineToPOrder(lRequestLineId As Long, lPOrderId As
```

```
Long, dQty As Double, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestLineId*: This parameter contains the identifier of the request line.
- *lPOrderId*: This parameter contains the identifier of the impacted purchase order.
- *dQty*: This parameter contains the quantity to order. In this way, you can limit the quantity orderd in relation to the quantity requested (in the unit of the product).
- *bCanMerge*: This parameter enables you to specify whether the line can be merged with an existing line in the purchase order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAddTemplateToPOrder()

This function enables you to add the full contents of a standard purchase order to a given purchase order.

API syntax

```
long AmAddTemplateToPOrder(handle hApiCnxBase, long lRequestId, long lPOrderId, long lTemplateId, long lQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddTemplateToPOrder(lRequestId As Long, lPOrderId As Long, lTemplateId As Long, lQty As Long, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestId*: This parameter contains the identifier of the request line to satisfy for the order lines that will be added.
- *lPOrderId*: This parameter contains the identifier of the impacted purchase order.
- *lTemplateId*: This parameter contains the identifier of the standard purchase order to add.
- *lQty*: This parameter contains the quantity (in the unit of the product) to add.

- *bCanMerge*: This parameter enables you to specify whether the line can be merged with an existing line in the purchase order.

Output parameters

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

AmAddTemplateToRequest()

This function enables you to add the full contents of a standard request to a given request.

API syntax

```
long AmAddTemplateToRequest(handle hApiCnxBase, long lReqId, long
lTemplateId, long lQty, long bCanMerge);
```

Internal Basic syntax

```
Function AmAddTemplateToRequest(lReqId As Long, lTemplateId As Long, lQty
As Long, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the affected request line.
- *lTemplateId*: This parameter contains the identifier of the standard request to add.
- *lQty*: This parameter contains the quantity (in the unit of the product) to add.
- *bCanMerge*: This parameter enables you to specify whether the line can be merged with an existing line in the request.

Output parameters

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

AmArchiveRecord()

This function archives a records in the database. The record is deleted from its original table and moved to the corresponding archival table.

API syntax

```
long AmArchiveRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmArchiveRecord(hApiRecord As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record concerned by the operation.

Output parameters

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

Notes

The processing of linked records depends on the type of the link. For OWN type links, linked records are processed identically. In the case of a DEFINE or NORMAL link, foreign keys of linked records are reset to 0 and the archival fields are populated with the identifier and description string of the archived record.

This function is available for a record from a standard table.

AmAttribCmdAvailability()

This function enables you to determine the availability of the Assign or Unassign button for a helpdesk ticket.

Internal Basic syntax

```
Function AmAttribCmdAvailability(bAttrib As Long, lGroupID As Long,
lInChargeID As Long, bInChargeIsReadOnly As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *bAttrib*: To test the availability of the Assign button, set this parameter to "1". To test for the availability of the Unassign button, set this parameter to "0".
- *lGroupID*: This parameter contains the identifier of the helpdesk group associated with the helpdesk group concerned.
- *lInChargeID*: This parameter contains the identifier of the ticket assignee.
- *bInChargeIsReadOnly*: This parameter is set to "1" if the assignee can only consult the ticket, "0" if they have modification rights.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmAuthenticate()

This function validates the user name and password.

Internal Basic syntax

```
Function AmAuthenticate(strUser As String, strPasswd As String) As Long
```

Field of application

Version: 3.0.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- `strUser: User name.`
- `strPasswd: Password.`

Output parameters

- -1: The user is unknown to all systems.
- 0: The user is known but the wrong password was provided.
- 1: The authentication is successful.

AmBackupRecord()

This function makes a backup copy of a record. The record is copied to the corresponding archival table without being deleted from its original table.

API syntax

```
long AmBackupRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmBackupRecord(hApiRecord As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record concerned by the operation.

Output parameters

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

Notes

The processing of linked records depends on the type of the link. For OWN type links, linked records are processed identically. In the case of a DEFINE or NORMAL link, foreign keys of linked records are reset to 0 and the archival fields are populated with the identifier and description string of the archived record.

This function is available for a record from a standard table.

AmBuildNumber()

This function returns the application's build number.

Internal Basic syntax

```
Function AmBuildNumber() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmBusinessSecondsInDay()

Calculates the number of business seconds in a day according to a calendar.

API syntax

```
long AmBusinessSecondsInDay(handle hApiCnxBase, char *strCalendarSqlName,
long tmDate);
```

Internal Basic syntax

```
Function AmBusinessSecondsInDay(strCalendarSqlName As String, tmDate As
Date) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strCalendarSqlName*: SQL name of the calendar used for the calculation.
- *tmDate*: Date for which the calculation is performed.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCalcConsolidatedFeature()

Calculates the value of consolidated feature on a table identified by its SQL name.

API syntax

```
long AmCalcConsolidatedFeature(handle hApiCnxBase, long lCalcFeatId, char *strSQLTableName);
```

Internal Basic syntax

```
Function AmCalcConsolidatedFeature(lCalcFeatId As Long, strSQLTableName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCalcFeatId*: Identifier of the consolidated feature.
- *strSQLTableName*: SQL name of the table for which the consolidated feature is calculated. The feature must be defined for this table.

Output parameters

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

AmCalcDepr()

This function enables you calculate the depreciation amount for an asset on a given date. It returns the depreciation value on this date.

API syntax

```
double AmCalcDepr(handle hApiCnxBase, long iType, long lDuration, double dCoeff, double dPrice, long tmStart, long tmDate);
```

Internal Basic syntax

```
Function AmCalcDepr(iType As Long, lDuration As Long, dCoeff As Double, dPrice As Double, tmStart As Date, tmDate As Date) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iType*: This parameter identifies the nature of the depreciation. The following values are possible:
  - 0: No depreciation
  - 1: Straight line
  - 2: Declining balance
- *lDuration*: This parameter contains the period over which the asset is depreciated. This period is expressed in seconds.
- *dCoeff*: This parameter contains the coefficient applied in the declining balance method. It is not interpreted in the case of the straight line depreciation method but must have a value.
- *dPrice*: This parameter contains the market value of the asset concerned by the depreciation calculation.
- *tmStart*: This parameter contains the date from which the asset is depreciated.
- *tmDate*: This parameter contains the date on which the depreciation and residual value of the asset are calculated.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCalculateAndStoreStatistic()

This function:

1. Calculates the statistic, identified by its primary key, using data from the database.
2. Returns a string containing the results of the statistic.
3. Stores the result in the **Statistics memos** (amStatMemo) table.

API syntax

```
long AmCalculateAndStoreStatistic(handle hApiCnxBase, long lStatisticId);
```

Internal Basic syntax

```
Function AmCalculateAndStoreStatistic(lStatisticId As Long) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lStatisticId*: Primary key of the statistic to be calculated and stored.

Output parameters

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

AmCalculateCatRefQty()

This function enables you to calculate the quantity of a catalog reference to be ordered to fulfil a purchase request.

API syntax

```
double AmCalculateCatRefQty(handle hApiCnxBase, long lSetQty, long
lUseUnitId, long lPurchUnitId, char *strModelDesc, char *strCatRefDesc,
char *strPurchUnit, char *strUseUnit, double dPkgQty, double dUnitConv,
double dReqLineQty);
```

Internal Basic syntax

```
Function AmCalculateCatRefQty(lSetQty As Long, lUseUnitId As Long,
lPurchUnitId As Long, strModelDesc As String, strCatRefDesc As String,
strPurchUnit As String, strUseUnit As String, dPkgQty As Double,
dUnitConv As Double, dReqLineQty As Double) As Double
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lSetQty*: This parameter contains the corresponding quantity of each item in the product (for example 1, in the case of a product containing 6 one-liter bottles of water).
- *lUseUnitId*: This parameter contains the identifier of the unit used for the model.
- *lPurchUnitId*: This parameter contains the identifier of the unit used for the product.
- *strModelDesc*: This parameter contains the description of the model.
- *strCatRefDesc*: This parameter contains the description of the catalog reference.
- *strPurchUnit*: This parameter contains the description of the unit used for the product.

- *strUseUnit*: This parameter contains the description of the unit used for the model.
- *dPkgQty*: This parameter contains the corresponding quantity of each item in the product (for example 1, in the case of a product containing 6 one-liter bottles of water).
- *dUnitConv*: This parameter contains the conversion ratio for units for the product.
- *dReqLineQty*: This parameter contains the quantity of the model stiplulated in the purchase request.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCalculateReqLineQty()

This function enables you to calculate the quantity of model required to create a purchase request.

API syntax

```
double AmCalculateReqLineQty(handle hApiCnxBase, long lSetQty, long
lUseUnitId, long lPurchUnitId, char *strModelDesc, char *strCatRefDesc,
char *strPurchUnit, char *strUseUnit, double dPkgQty, double dUnitConv,
double dCatRefQty);
```

Internal Basic syntax

```
Function AmCalculateReqLineQty(lSetQty As Long, lUseUnitId As Long,
lPurchUnitId As Long, strModelDesc As String, strCatRefDesc As String,
strPurchUnit As String, strUseUnit As String, dPkgQty As Double,
dUnitConv As Double, dCatRefQty As Double) As Double
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lSetQty*: This parameter contains the corresponding quantity of each item in the product (for example 1, in the case of a product containing 6 one-liter bottles of water).
- *lUseUnitId*: This parameter contains the identifier of the unit used for the model.
- *lPurchUnitId*: This parameter contains the identifier of the unit used for the product.
- *strModelDesc*: This parameter contains the description of the model.
- *strCatRefDesc*: This parameter contains the description of the catalog reference.
- *strPurchUnit*: This parameter contains the description of the unit used for the product.
- *strUseUnit*: This parameter contains the description of the unit used for the model.
- *dPkgQty*: This parameter contains the corresponding quantity of each item in the product (for example 1, in the case of a product containing 6 one-liter bottles of water).
- *dUnitConv*: This parameter contains the conversion ratio for units for the product.
- *dCatRefQty*: This parameter contains the quantity of the model stiplulated in the ordered catalog reference.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCalculateStatistic()

This function calculates the statistic identified by its primary key.

- If the statistic was stored in the **Statistics memos** (amStatMemo) table via the `AmCalculateAndStoreStatistic` function, then the function calculates the statistic from the data stored in this table (function executes more rapidly).
- Otherwise, the function calculates the statistic using the data directly from the database (function executes more slowly).

Then the function returns a string containing the results of the statistic.

API syntax

```
long AmCalculateStatistic(handle hApiCnxBase, long lStatisticId, char
*pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmCalculateStatistic(lStatisticId As Long) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lStatisticId*: Primary key of the statistic to be calculated.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCalculateStatisticFromSQLName()

This function calculates the statistic identified by its SQL name.

- If the statistic was stored in the **Statistics memos** (amStatMemo) table via the `AmCalculateAndStoreStatistic` function, then the function calculates the statistic from the data stored in this table (function executes more rapidly).
- Otherwise, the function calculates the statistic using the data directly from the database (function executes more slowly).

Then the function returns a string containing the results of the statistic.

API syntax

```
long AmCalculateStatisticFromSQLName(handle hApiCnxBase, char
*strSqlName, char *pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmCalculateStatisticFromSQLName(strSqlName As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strSqlName* : SQL name of the statistic to be calculated.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCbkReplayEvent()

This function enables you to replay the chargeback rule at the origin of an event, after correcting the record at the origin of the event.

API syntax

```
long AmCbkReplayEvent(handle hApiCnxBase, long lCbkEventId);
```

Internal Basic syntax

```
Function AmCbkReplayEvent(lCbkEventId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCbkEventId*: This parameter contains the identifier of the chargeback event concerned.

Output parameters

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

AmCheckTraceDone()

The AmCheckTraceDone API determines if a port (lPortId) or bundle (lBundleId) is connected to an existing trace. The trace direction (iTraceDir) identifies if the trace should be checked in the direction of user-to-host (iTraceDir = 1) or host-to-user (iTraceDir = 0).

API syntax

```
long AmCheckTraceDone(handle hApiCnxBase, long lPortId, long lBundleId,
```

```
long iTraceDir);
```

Internal Basic syntax

```
Function AmCheckTraceDone(lPortId As Long, lBundleId As Long, iTraceDir
As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lPortId*: This parameter is the port ID to check.
- *lBundleId*: This parameter is the bundle ID to check.
- *iTraceDir*: This parameter defines the direction to check.
  - 1: Check in the host direction
  - 0: Check in the host direction

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCleanup()

This function must be named at the end of scripts using the database modification functions. It frees all used resources.

API syntax

```
void AmCleanup();
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

AmClearLastError()

This function clears the information concerning the last error message occurred during the last function call.

API syntax

```
long AmClearLastError(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmClearLastError() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

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

AmClientType()

This function returns the numerical identifier of the client types listed as follows:

- CLIENTTYPE_UNKNOWN: 0
- CLIENTTYPE_CLIENT: 1
- CLIENTTYPE_API: 2
- CLIENTTYPE_WEBSERVICES: 3

Internal Basic syntax

```
Function AmClientType () As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the **AmLastError()** function (and optionally the **AmLastErrorMsg()** function) to find out if an error occurred (and obtain its associated message).

AmCloseAllChildren()

This function destroys all the objects created during the current connection.

API syntax

```
long AmCloseAllChildren(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmCloseAllChildren() As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

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

AmCloseConnection()

Ends the Asset Manager session for a given connection. All objects (queries, records, tables, fields, and so on) created within this connection are automatically destroyed. Their handles become invalid. The connection handle no longer exists.

API syntax

```
long AmCloseConnection(handle hApiCnxBase);
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

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

AmCommit()

This function commits all the modifications made to the database associated with the connection.

API syntax

```
long AmCommit(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmCommit() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmComputeAllLicAndInstallCounts()

This function performs the count of software licenses and installations for all records.

API syntax

```
long AmComputeAllLicAndInstallCounts(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmComputeAllLicAndInstallCounts() As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

Notes

This function does not work with the Software License Optimization best practice package.

AmComputeLicAndInstallCounts()

This function performs the count of software licenses and installations for a record.

API syntax

```
long AmComputeLicAndInstallCounts(handle hApiCnxBase, long lSLCountId);
```

Internal Basic syntax

```
Function AmComputeLicAndInstallCounts(lSLCountId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lSLCountId*: This parameter contains the identifier of the software license counter.

Output parameters

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

AmConnectionName()

This function returns the current database connection name.

API syntax

```
long AmConnectionName(handle hApiCnxBase, char *return, long lreturn);
```

Internal Basic syntax

```
Function AmConnectionName() As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal=amConnectionName()
```

AmConnectTrace()

The AmConnectTrace API is for connecting a source device/cable to a destination device/cable and creating a trace history and a trace operation.

API syntax

```
long AmConnectTrace(handle hApiCnxBase, long iSrcLinkType, long
lSrcPortBunId, long lSrcLabelRuleId, long iDestLinkType, long
lDestPortBunId, long lDestLabelRuleId, long iTraceDir, long lDutyId, char
*strComment, long lCabTraceOutId);
```

Internal Basic syntax

```
Function AmConnectTrace(iSrcLinkType As Long, lSrcPortBunId As Long,
lSrcLabelRuleId As Long, iDestLinkType As Long, lDestPortBunId As Long,
lDestLabelRuleId As Long, iTraceDir As Long, lDutyId As Long, strComment
As String, lCabTraceOutId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *iSrcLinkType*: This parameter determines the trace type for the source device/cable.
  - 8: Cable
  - 9: Device
- *lSrcPortBunId*: This parameter is the port or bundle to be connected on the source side.
- *lSrcLabelRuleId*: This parameter is the label rule used for the source link.
- *iDestLinkType*: This parameter determines the trace type for the destination device/cable.
  - 8: Cable
  - 9 Device
- *lDestPortBunId*: This parameter is the port or bundle to be connected on the destination side.
- *lDestLabelRuleId*: This parameter is the label rule used for the destination link.
- *iTraceDir*: This parameter defines the direction of the connection.
  - 1: user to host
  - 0: host to user
- *lDutyId*: This parameter is the duty for a cable type link.
- *strComment*: This parameter is the label for the trace operation.
- *lCabTraceOutId*: This parameter is the Cable Trace Output ID.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertCurrency()

This function performs a conversion between two currencies at a given date.

API syntax

```
double AmConvertCurrency(handle hApiCnxBase, long tmDate, char
*strSrcName, char *strDstName, double dVal);
```

Internal Basic syntax

```
Function AmConvertCurrency(tmDate As Date, strSrcName As String,
strDstName As String, dVal As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *tmDate*: This parameter contains the conversion date. It enables you to know the conversion rate in effect on this date.
- *strSrcName*: This parameter contains the source currency for the conversion, that is, the currency you want to convert.
- *strDstName*: This parameter contains the target currency for the conversion, that is, the currency in which you want to express the source currency.
- *dVal*: This parameter contains the amount (expressed in the monetary unit of the source currency) to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The currency parameters (`strSrcName` and `strDstName`) for this function mustbe defined in Asset Manager. Furthermore, a valid exchange rate must exist for the date when you want to perform the conversion (tmDate parameter).

Example

The following example converts 5,000 FRF into dollars, on the date of November 02, 1998.

```
AmConvertCurrency("1998/11/02 00:00:00", "FRF", "$", 5000)
```

AmConvertDateBasicToUnix()

This function converts a Basic format date ("Date" type) to a Unix format date ("Long" type). This function does not work from external tools because the two types are equivalent.

API syntax

```
long AmConvertDateBasicToUnix(handle hApiCnxBase, long tmTime);
```

Internal Basic syntax

```
Function AmConvertDateBasicToUnix(tmTime As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *tmTime*: This parameter contains the date to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDateIntlToUnix()

This function converts an international format date ("Date" type) to a Unix format date ("Long" type).

API syntax

```
long AmConvertDateIntlToUnix(handle hApiCnxBase, char *strDate);
```

Internal Basic syntax

```
Function AmConvertDateIntlToUnix(strDate As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDate*: This parameter contains the date to be converted in the international format (yyyy-mm-dd hh:mm:ss).

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDateStringToUnix()

Converts a date to a string (as displayed in the Windows Control Panel) to a Unix "Long".

API syntax

```
long AmConvertDateStringToUnix(handle hApiCnxBase, char *strDate);
```

Internal Basic syntax

```
Function AmConvertDateStringToUnix(strDate As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDate*: Date as string to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDateUnixToBasic()

This function converts a Unix format date ("Long" type) to a Basic format date ("Date" type). This function does not work from external tools because the two types are equivalent.

API syntax

```
long AmConvertDateUnixToBasic(handle hApiCnxBase, long lTime);
```

Internal Basic syntax

```
Function AmConvertDateUnixToBasic(lTime As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lTime*: This parameter contains the date to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDateUnixToIntl()

This function converts a Unix format date ("Long" type) to an international format date (yyyy-mm-dd hh:mm:ss).

API syntax

```
long AmConvertDateUnixToIntl(handle hApiCnxBase, long lUnixDate, char *pstrDate, long lDate);
```

Internal Basic syntax

```
Function AmConvertDateUnixToIntl(lUnixDate As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`lUnixDate`*: This parameter contains the date to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDateUnixToString()

Converts a "Long" Unix format date to a string format date (as displayed in the Windows Control Panel).

API syntax

```
long AmConvertDateUnixToString(handle hApiCnxBase, long lUnixDate, char *pstrDate, long lDate);
```

Internal Basic syntax

```
Function AmConvertDateUnixToString(lUnixDate As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lUnixDate*: "Long" Unix format date to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertDoubleToString()

This function converts a double precision number to a string. The string is formatted according to the regional options (number) defined in the Windows Control Panel.

API syntax

```
long AmConvertDoubleToString(double dSrc, char *pstrDst, long lDst);
```

Internal Basic syntax

```
Function AmConvertDoubleToString(dSrc As Double) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *dSrc*: This parameter contains the double-precision number to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertMonetaryToString()

This function converts a monetary value to a character string. The string is formatted according to the regional options (currency) defined in the Windows Control Panel.

API syntax

```
long AmConvertMonetaryToString(double dSrc, char *pstrDst, long lDst);
```

Internal Basic syntax

```
Function AmConvertMonetaryToString(dSrc As Double) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *dSrc*: This parameter contains the monetary value you want to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertStringToDouble()

This function converts a character string (in a format corresponding to the one defined in the Windows Control Panel) to a double precision number.

API syntax

```
double AmConvertStringToDouble(char *strSrc);
```

Internal Basic syntax

```
Function AmConvertStringToDouble(strSrc As String) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strSrc*: This parameter contains the character string to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmConvertStringToMonetary()

This function converts a character string (in a format corresponding to the one defined in the Windows Control Panel) to a monetary value.

API syntax

```
double AmConvertStringToMonetary(char *strSrc);
```

Internal Basic syntax

```
Function AmConvertStringToMonetary(strSrc As String) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strSrc*: This parameter contains the character string to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCounter()

This function returns the value of the *strCounterName* counter, incremented by 1. Zeros are added as padding at the beginning if *iWidth* is greater than the number of digits of the counter. If the counter has more digits than the value stored in *iWidth*, the result will not be truncated.

Internal Basic syntax

```
Function AmCounter(strCounterName As String, iWidth As Long) As String
```

Field of application

Version: 2.52

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strCounterName*: Name of the counter as it is defined in Asset Manager (access via the **Administration/ Counters** menu item).
- *iWidth*: The value of this parameter forces the output format of the function to be expressed over n digits. This parameter is only useful if the size of the counter is less than the value of this parameter.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If this function is used in a Script type action, you must specify a context for the action. Otherwise, an error is generated.

Example

The following example returns the value of the "Delivery" counter expressed in 5 digits:

```
Dim strCounterName As String
strCounter = AmCounter("Delivery", 5)
```

For example, if the "Delivery" counter equals "18", the function returns:

```
00019
```

AmCreateAssetPort()

The AmCreateAssetPort API creates a new port on a device (lAssetId). The new port contains the given number of pins (iPinCount) of the given cable connector type (lCabCnxTypeId). The status of the pins must be "Available". The pins that will be added to the port are sorted by sequence number. Depending on the port direction (iPinPortDir), the available pins are sorted in ascending (iPinPortDir = 0) or descending (iPinPortDir = 1) order. This function assigns the given duty (lDutyId) to the new port.

API syntax

```
long AmCreateAssetPort(handle hApiCnxBase, long lAssetId, long
lCabCnxTypeId, long lDutyId, long iPinCount, long bPinPortDir, long
iConnStatus, long bConsecutivePins, long iPrevPinSeq, long bLogError);
```

Internal Basic syntax

```
Function AmCreateAssetPort(lAssetId As Long, lCabCnxTypeId As Long,
lDutyId As Long, iPinCount As Long, bPinPortDir As Long, iConnStatus As
Long, bConsecutivePins As Long, iPrevPinSeq As Long, bLogError As Long)
As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lAssetId*: This parameter is the device ID.
- *lCabCnxTypeId*: This parameter is the cable connection type ID.
- *lDutyId*: This parameter is the duty type ID of the port.
- *iPinCount*: This parameter is the pin count that will be used in the new port.
- *bPinPortDir*: This parameter specifies the direction of the port.
- *iConnStatus*
- *bConsecutivePins*
- *iPrevPinSeq*
- *bLogError*

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateAssetsAwaitingDelivery()

This function enables you to create the assets that are awaiting receipt

API syntax

```
long AmCreateAssetsAwaitingDelivery(handle hApiCnxBase, long lPOrdId);
```

Internal Basic syntax

```
Function AmCreateAssetsAwaitingDelivery(lPOrdId As Long) As Long
```

Field of application

Version: 3.61

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdId*: This parameter contains the identifier of the purchase order concerned

Output parameters

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

AmCreateCable()

The AmCreateCable API creates a new cable. The cable is created using the given model type (lModelId), the role of the cable (strCableRole), its label rule (lLabelRuleId), its user location (lUserLoc), and its host location (lHostLoc). If the project (lProjectId) and work order (lWorkOrderId) have values,

the new cable is added to the project and work order with the given comment (strComment). This comment describes the action that will be performed on the cable (that is, "Install new cable").

API syntax

```
long AmCreateCable(handle hApiCnxBase, long lModelId, long lUserId, long lHostId, char *strCableRole, long lProjectId, long lWorkOrderId, char *strComment, long lLabelRuleId, char *strLabel);
```

Internal Basic syntax

```
Function AmCreateCable(lModelId As Long, lUserId As Long, lHostId As Long, strCableRole As String, lProjectId As Long, lWorkOrderId As Long, strComment As String, lLabelRuleId As Long, strLabel As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lModelId*: This parameter is the cable model ID.
- *lUserId*: This parameter is the user side location ID.
- *lHostId*: This parameter is the host side location ID.
- *strCableRole*: This parameter defines the role of the cable.
- *lProjectId*: This parameter defines the project associated with the placement of the cable.
- *lWorkOrderId*: This parameter defines the work order associated with the placement of the cable.

- *strComment*: This parameter is the comment used on the work order (defined by lWorkOrderId).
- *lLabelRuleId*: This parameter defines the label rule that will be applied to create the label for the cable.
- *strLabel*: This parameter specifies the label affixed to the cable.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateCableBundle()

The AmCreateCableBundle API creates a new bundle on a cable (lCableId). The new bundle contains the given number of cable pairs (iPairCount) of the given cable pair type (lPairType). The status of the pairs must be "Available". This function assigns the given duty (lDutyId) to the new bundle.

API syntax

```
long AmCreateCableBundle(handle hApiCnxBase, long lCableId, long
lPairTypeId, long lDutyId, long iPairCount, long iStartPairSeq, long
bLogError);
```

Internal Basic syntax

```
Function AmCreateCableBundle(lCableId As Long, lPairTypeId As Long,
lDutyId As Long, iPairCount As Long, iStartPairSeq As Long, bLogError As
Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the cable ID (it must exist in the cable table).
- *lPairTypeId*: This parameter is the cable pair type ID.
- *lDutyId*: This parameter is the duty of the cable bundle ID.
- *iPairCount*This parameter defines the pair count of the bundle.
- *iStartPairSeq*
- *bLogError*

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateCableLink()

The AmCreateCableLink API creates a new cable type cable link for a given cable (lCableId) and bundle (lNextBundle). The duty of the cable link is set using the given duty (lDutyId). The label rule of the cable link is set using the given label rule (lLabelRule).

> The label is not updated using the given label rule, a separate call must be made toAmRefreshLabel().

If a previous link (lPrevLinkId) is specified, a parent link is made between the two records where the previous link is the child.

API syntax

```
long AmCreateCableLink(handle hApiCnxBase, long lCableId, long lDutyId,
long lBundleId, long lPrevLinkId, long iTraceDir, long lLabelRuleId);
```

Internal Basic syntax

```
Function AmCreateCableLink(lCableId As Long, lDutyId As Long, lBundleId
As Long, lPrevLinkId As Long, iTraceDir As Long, lLabelRuleId As Long) As
Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the cable ID for the connection.
- *lDutyId*: This parameter is the connection duty.
- *lBundleId*: This parameter is the ID of the cable bundle to connect.
- *lPrevLinkId*: This parameter defines the cable link ID used to connect. This is optional by using a value of 0.

- *iTraceDir*: This parameter defines the connection direction.
  - 0=host to user
  - 1=user to host
- *lLabelRuleId*: This parameter is the label rule ID used.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateDelivFromPO()

This function receives a purchase order and returns the identifier of the receiving slip created.

API syntax

```
long AmCreateDelivFromPO(handle hApiCnxBase, long lPOrdId);
```

Internal Basic syntax

```
Function AmCreateDelivFromPO(lPOrdId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdId*: This parameter contains the identifier of the purchase order to be received.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateDevice()

The AmCreateDevice API creates a new device. The device is created using the given model type (lProductId) and location (lLocId). The label rule of the asset is set to the given rule (lLabelRuleId).

> The label is not updated using the given label rule, a separate call must be made to AmRefreshLabel.

If the project (lProjectId) and work order (lWorkOrderId) have values, the new asset is added to the project and work order with the comment contained in strComment. This comment describes the action that will be performed on the asset (that is, "Install new asset").

API syntax

```
long AmCreateDevice(handle hApiCnxBase, long lModelId, long lLocationId,
long lProjectId, long lWorkOrderId, long lLabelRuleId, char *strComment);
```

Internal Basic syntax

```
Function AmCreateDevice(lModelId As Long, lLocationId As Long, lProjectId
As Long, lWorkOrderId As Long, lLabelRuleId As Long, strComment As
String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lModelId*: This parameter is the model ID for the new device.
- *lLocationId*: This parameter is the location ID for the new device.
- *lProjectId*: The parameter is the project ID. It can be 0.
- *lWorkOrderId*: This parameter is the work order ID. It can be 0.
- *lLabelRuleId*: This parameter defines the label rule ID that will be used for the asset.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateDeviceLink()

The AmCreateDeviceLink API creates a new device type cable link for a given device (lAssetId) and port (lPortId). The label rule of the cable link is set using the given label rule (lLabelRule).

> The label is not updated using the given label rule, a separate call must be made to AmRefreshLabel.

If a previous link (lPrevLinkId) is specified, a parent link is made between the two records. If the trace direction is user-to-host (iTraceDir = 1), then the previous link is the child. If the trace direction is host-to-user (iTraceDir = 0) then the previous link is the parent.

API syntax

```
long AmCreateDeviceLink(handle hApiCnxBase, long lAssetId, long lPortId,
long lPrevLinkId, long iTraceDir, long lLabelRuleId);
```

Internal Basic syntax

```
Function AmCreateDeviceLink(lAssetId As Long, lPortId As Long,
lPrevLinkId As Long, iTraceDir As Long, lLabelRuleId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lAssetId*: This parameter contains the identifier of the asset that will be connected.
- *lPortId*: This parameter contains the identifier of the port that will be connected.
- *lPrevLinkId*: This parameter contains the identifier of the device link enabling the connection.
- *iTraceDir*: This parameter specifies the direction of the connection.
  - 0=host to user
  - 1=user to host
- *lLabelRuleId*: This parameter contains the identifier of the label rule used for the new connection.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateEstimFromReq()

This function creates an estimate from a purchase request and returns the identifier of the estimate created.

API syntax

```
long AmCreateEstimFromReq(handle hApiCnxBase, long lReqId, long lSuppId);
```

Internal Basic syntax

```
Function AmCreateEstimFromReq(lReqId As Long, lSuppId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the purchase request used to create the estimate.
- *lSuppId*: This parameter contains the identifier of the supplier of the estimate that will be created by the function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateEstimsFromAllReqLines()

This function creates an estimate from a request and returns the identifier of the estimate created.

API syntax

```
long AmCreateEstimsFromAllReqLines(handle hApiCnxBase, long lReqId, long
```

```
bMergeLines, long lDefSuppId);
```

Internal Basic syntax

```
Function AmCreateEstimsFromAllReqLines(lReqId As Long, bMergeLines As
Long, lDefSuppId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the request at the origin of the estimate.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.
- *lDefSuppId*: This parameter contains the identifier of the default supplier for the estimate.

Output parameters

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

AmCreateInvFromPO()

This function creates a supplier invoice from a purchase order and returns the identifier of the supplier invoice created.

API syntax

```
long AmCreateInvFromPO(handle hApiCnxBase, long lPOrdId);
```

Internal Basic syntax

```
Function AmCreateInvFromPO(lPOrdId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdId*: This parameter contains the identifier of the purchase order at the origin of the invoice.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateLink()

This function modifies a link of a record and makes it point to a new record (*hApiRecDest*) in the target table. It therefore creates a link between two records.

API syntax

```
long AmCreateLink(handle hApiRecord, char *strLinkName, handle
hApiRecDest);
```

Internal Basic syntax

```
Function AmCreateLink(hApiRecord As Long, strLinkName As String,
hApiRecDest As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record containing the link to be modified.
- *strLinkName*: This parameter contains the SQL name of the link to be modified.

- *hApiRecDest*: This parameter contains a handle of the target record of the link.

Output parameters

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

AmCreateOrUpdateInvoiceFromReceipt()

This function enables you to create or update an invoice from a receiving slip.

API syntax

```
long AmCreateOrUpdateInvoiceFromReceipt(handle hApiCnxBase, long
lRecptId);
```

Internal Basic syntax

```
Function AmCreateOrUpdateInvoiceFromReceipt(lRecptId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lRecptId*: This parameter contains the identifier of the invoice concerned by the operation.

Output parameters

The function returns the identifier of the generated invoice.

Notes

Update is not possible by calling this function from an external tool.

AmCreatePOFromEstim()

This function creates a purchase order from an estimate and returns the identifier of the purchase order created.

API syntax

```
long AmCreatePOFromEstim(handle hApiCnxBase, long lEstimId);
```

Internal Basic syntax

```
Function AmCreatePOFromEstim(lEstimId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lEstimId*: This parameter contains the identifier of the estimate used to create the order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreatePOFromReq()

This function creates a purchase order from a purchase request and returns the identifier of the PO created.

API syntax

```
long AmCreatePOFromReq(handle hApiCnxBase, long lReqId, long lSuppId);
```

Internal Basic syntax

```
Function AmCreatePOFromReq(lReqId As Long, lSuppId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the purchase request used to create the purchase order.
- *lSuppId*: This parameter contains the identifier of the supplier of the purchase order that will be created by the function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreatePOrderFromRequest()

This function enables you to create a purchase order from a request.

API syntax

```
long AmCreatePOrderFromRequest(handle hApiCnxBase, long lRequestId, long lSupplierId);
```

Internal Basic syntax

```
Function AmCreatePOrderFromRequest(lRequestId As Long, lSupplierId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestId*: This parameter contains the identifier of the request concerned.
- *lSupplierId*: This parameter contains the identifier of the supplier for the purchase order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreatePOrdersFromRequest()

This function enables you to create all the purchase orders necessary to satisfy a given request.

API syntax

```
long AmCreatePOrdersFromRequest(handle hApiCnxBase, long lRequestId);
```

Internal Basic syntax

```
Function AmCreatePOrdersFromRequest(lRequestId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestId*: This parameter contains the identifier of the request concerned

Output parameters

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

AmCreatePOsFromAllReqLines()

This function creates all the purchase orders from the request lines of a request.

API syntax

```
long AmCreatePOsFromAllReqLines(handle hApiCnxBase, long lReqId, long
bMergeLines, long lDefSuppId);
```

Internal Basic syntax

```
Function AmCreatePOsFromAllReqLines(lReqId As Long, bMergeLines As Long,
lDefSuppId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lReqId*: This parameter contains the identifier of the request from which the purchase orders are to be created.
- *bMergeLines*: This parameter enables you to specify if identical request lines are to be combined (*bMergeLines*=1) to given one single line. The quantities given for the lines to be combined are added together and a single line is created.
- *lDefSuppId*: This parameter contains the identifier of the default supplier for the requested items. This parameter is optional and is set to "0" by default.

Output parameters

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

AmCreateProjectCable()

The AmCreateProjectCable API adds a cable (lCableId) to a project (lProjectId) and work order (lWorkOrderId). A comment (strComment) explains the action being performed (that is, "Install new cable").

API syntax

```
long AmCreateProjectCable(handle hApiCnxBase, long lProjectId, long lWorkOrderId, long lCableId, char *strComment);
```

Internal Basic syntax

```
Function AmCreateProjectCable(lProjectId As Long, lWorkOrderId As Long, lCableId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lProjectId*: This parameter is the ID of the project that gets the cable.
- *lWorkOrderId*: This parameter is the ID of the work order for the cable.
- *lCableId*: This parameter is the cable ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateProjectDevice()

The AmCreateProjectDevice API adds a device (lAssetId) to a project (lProjectId) and work order (lWorkOrderId). A comment (strComment) explains the action being performed (that is, "Install new device").

API syntax

```
long AmCreateProjectDevice(handle hApiCnxBase, long lProjectId, long
lWorkOrderId, long lAssetId, char *strComment);
```

Internal Basic syntax

```
Function AmCreateProjectDevice(lProjectId As Long, lWorkOrderId As Long,
lAssetId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓

	Available
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lProjectId*: This parameter defines the ID of the project to get the new device.
- *lWorkOrderId*: This parameter defines the ID of the work order to get the new device.
- *lAssetId*: This parameter is the new device asset ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateProjectTrace()

The AmCreateProjectTrace API adds a trace (strTrace) to a project (lProjectId) and work order (lWorkOrderId). The service of the trace is set using the given duty (lDutyId). The trace type (lTraceType) indicates if the trace is a connection (iTraceType = 1) or a disconnection (lTraceType = 2). The label of the user link being modified (strModLinkLabel) identifies what part of the trace is being modified. A comment (strComment) explains the action being performed (that is, "Connect these devices").

API syntax

```
long AmCreateProjectTrace(handle hApiCnxBase, long lProjectId, long
lWorkOrderId, long iTraceType, long lDutyId, char *strModLinkLabel, char
```

```
*strTrace, char *strComment);
```

Internal Basic syntax

```
Function AmCreateProjectTrace(lProjectId As Long, lWorkOrderId As Long,
iTraceType As Long, lDutyId As Long, strModLinkLabel As String, strTrace
As String, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lProjectId*: This parameter defines the project ID to get the trace information.
- *lWorkOrderId*: This parameter defines the work order ID to get the trace information.
- *iTraceType*: This parameter defines the trace type.
  - 1=connection
  - 2=disconnection
- *lDutyId*: This parameter defines the duty. This appears in the work order.
- *strModLinkLabel*: This parameter defines a comment that will be used on the work order.
- *strTrace*: This parameter defines the trace output string that will be used on the work order.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateReceiptFromPOrder()

This function enables you to create a receipt from a purchase order.

API syntax

```
long AmCreateReceiptFromPOrder(handle hApiCnxBase, long lPOrderId);
```

Internal Basic syntax

```
Function AmCreateReceiptFromPOrder(lPOrderId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrderId*: This parameter contains the identifier of the purchase order concerned.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateRecord()

This function creates an empty record in a table taking the default values into account. This new record does not exist in the database until it has been inserted.

API syntax

```
handle AmCreateRecord(handle hApiCnxBase, char *strTable);
```

Internal Basic syntax

```
Function AmCreateRecord(strTable As String) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strTable*: This parameter contains the SQL name of the table in which you want to create the record.

Example

The following example creates an employee in the database:

```
Dim lErr As Long
Dim hRecord As Long
hRecord = amCreateRecord("amEmplDept")
lErr = amSetFieldStrValue(hRecord, "Name", "Doe")
lErr = amSetFieldStrValue(hRecord, "FirstName", "John")
lErr = amInsertRecord(hRecord)
```

AmCreateRequestToInvoice()

This function enables you to create all objects in the procurement cycle: Request, Purchase order, Receipt, Invoice.

API syntax

```
long AmCreateRequestToInvoice(handle hApiCnxBase, double dQty, long
lCatRefId, double dUnitPrice, char *strCur, long lRequesterId, long
lCostId, long lUserId, long lStockId);
```

Internal Basic syntax

```
Function AmCreateRequestToInvoice(dQty As Double, lCatRefId As Long,
dUnitPrice As Double, strCur As String, lRequesterId As Long, lCostId As
Long, lUserId As Long, lStockId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *dQty*: This parameter contains the quantity (in packaged units) to be ordered, then received, then invoiced.
- *lCatRefId*: This parameter contains the identifier of the catalog reference.
- *dUnitPrice*: This parameter contains the unit price of the catalog reference.
- *strCur*: This parameter contains the currency code for the catalog reference price.
- *lRequesterId*: This parameter contains the identifier of the requester.
- *lCostId*: This parameter contains the identifier of the impacted cost center.
- *lUserId*: This parameter contains the identifier of the user of the ordered item.
- *lStockId*: This parameter contains the identifier of the delivery stock of the item.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

Equivalent to the sequence of calls: amCreateRequestToReceipt, amCreateOrUpdateInvoiceFromReceipt.

AmCreateRequestToPOrder()

This function enables you to create the objects in the procurement cycle: Request, Purchase order.

API syntax

```
long AmCreateRequestToPOrder(handle hApiCnxBase, double dQty, long
lCatRefId, double dUnitPrice, char *strCur, long lRequesterId, long
lCostId, long lUserId, long lStockId);
```

Internal Basic syntax

```
Function AmCreateRequestToPOrder(dQty As Double, lCatRefId As Long,
dUnitPrice As Double, strCur As String, lRequesterId As Long, lCostId As
Long, lUserId As Long, lStockId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *dQty*: This parameter contains the quantity (in packaged units) to be ordered.
- *lCatRefId*: This parameter contains the identifier of the catalog reference.
- *dUnitPrice*: This parameter contains the unit price of the catalog reference.
- *strCur*: This parameter contains the currency code for the catalog reference price.

- *lRequesterId*: This parameter contains the identifier of the requester.
- *lCostId*: This parameter contains the identifier of the impacted cost center.
- *lUserId*: This parameter contains the identifier of the user of the ordered item.
- *lStockId*: This parameter contains the identifier of the delivery stock of the item.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateRequestToReceipt()

This function enables you to create the objects in the procurement cycle: Request, Purchase order, Receipt.

API syntax

```
long AmCreateRequestToReceipt(handle hApiCnxBase, double dQty, long
lCatRefId, double dUnitPrice, char *strCur, long lRequesterId, long
lCostId, long lUserId, long lStockId);
```

Internal Basic syntax

```
Function AmCreateRequestToReceipt(dQty As Double, lCatRefId As Long,
dUnitPrice As Double, strCur As String, lRequesterId As Long, lCostId As
Long, lUserId As Long, lStockId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *dQty*: This parameter contains the quantity (in packaged units) to be ordered, then received.
- *lCatRefId*: This parameter contains the identifier of the catalog reference.
- *dUnitPrice*: This parameter contains the unit price of the catalog reference.
- *strCur*: This parameter contains the currency code for the catalog reference price.
- *lRequesterId*: This parameter contains the identifier of the requester.
- *lCostId*: This parameter contains the identifier of the impacted cost center.
- *lUserId*: This parameter contains the identifier of the user of the ordered item.
- *lStockId*: This parameter contains the identifier of the delivery stock of the item.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

Equivalent to the sequence of calls: amCreateRequestToPOrder, amCreateReceiptFromPOrder.

AmCreateReturnFromReceipt()

This function enables you to create a return slip from a receiving slip.

API syntax

```
long AmCreateReturnFromReceipt(handle hApiCnxBase, long lRecptId);
```

Internal Basic syntax

```
Function AmCreateReturnFromReceipt(lRecptId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRecptId*: This parameter contains the identifier of the receipt line.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCreateTraceHist()

The AmCreateTraceHist API is for creating trace history and trace operation based on an existing connection from a source device/cable to a destination device/cable.

API syntax

```
long AmCreateTraceHist(handle hApiCnxBase, long lSrcLinkId, long
lDestLinkId, long iTraceDir, long lCabTraceOutId, char *strComment);
```

Internal Basic syntax

```
Function AmCreateTraceHist(lSrcLinkId As Long, lDestLinkId As Long,
iTraceDir As Long, lCabTraceOutId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lSrcLinkId*: This parameter is the device/cable used for the source link.
- *lDestLinkId*: This parameter is the device/cable used for the destination link.

- *iTraceDir*: This parameter specifies the direction of the connection.
  - 0=host to user
  - 1=user to host
- *lCabTraceOutId*: This parameter is the cable trace output ID.
- *strComment*: This parameter is the comment to be associated with the trace operation.

Output parameters

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

AmCreateTraceLink()

This function enables you to create a link between cable devices.

Internal Basic syntax

```
Function AmCreateTraceLink(iLinkType As Long, lAstCabId As Long,
lPrtBunId As Long, lPrevLinkId As Long, iTraceDir As Long, lDutyId As
Long, lLabelRuleId As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *iLinkType*: This parameter enables you to identify the type of element taken into account ("1" for a cable device, "0" for a cable).
- *lAstCabId*: This parameter contains the identifier of the asset associated with the cable device.
- *lPrtBunId*: This parameter contains the identifier of the record concerned by the operation. This identifier is taken in the amCableBundle table for a cable or in the amPort table for a cable device.
- *lPrevLinkId*: This parameter contains the identifier of the element used as starting point for the link.
- *iTraceDir*: This parameter enables you to specify the direction of the link. Either "HOST_TO_USER" or "USER_TO_HOST".
- *lDutyId*: This parameter contains the identifier of the link duty.
- *lLabelRuleId*: This parameter contains the identifier of the label rule of the link (by default, this value is null).

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCryptPassword()

This function encrypts the password of a user, identified by a login and password.

API syntax

```
long AmCryptPassword(handle hApiCnxBase, char *strUser, char *strPasswd,
char *pStrCrypted, long lpStrCrypted);
```

Internal Basic syntax

```
Function AmCryptPassword(strUser As String, strPasswd As String) As
String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strUser*: This parameter contains the login of the user whose password you want to encrypt.
- *strPasswd*: This parameter contains, in plaintext, the password to be encrypted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCurrentDate()

This function returns the current date on the client workstation.

API syntax

```
long AmCurrentDate();
```

Internal Basic syntax

```
Function AmCurrentDate() As Date
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the database is configured to use time zones, the behavior of this function differs depending on whether it is named directly from AssetCenter or from an external program. In AssetCenter, this function behaves in the same way as the Now() function in Basic. From external programs, the value returned by this function is expressed as GMT+0 and does not take daylight savings into account.

AmCurrentIsoLang()

This function returns the ISO language code of the language used in Asset Manager ("en" for English, "fr" for French, and so on).

API syntax

```
long AmCurrentIsoLang(char *pstrLanguage, long lLanguage);
```

Internal Basic syntax

```
Function AmCurrentIsoLang() As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCurrentLanguage()

This function returns the language version of Asset Manager ("US" for English, "FR" for French, and so on).

API syntax

```
long AmCurrentLanguage(char *pstrLanguage, long lLanguage);
```

Internal Basic syntax

```
Function AmCurrentLanguage() As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmCurrentServerDate()

This function returns the current date on the server.

API syntax

```
long AmCurrentServerDate(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmCurrentServerDate() As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDateAdd()

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

API syntax

```
long AmDateAdd(long tmStart, long tsDuration);
```

Internal Basic syntax

```
Function AmDateAdd(tmStart As Date, tsDuration As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example illustrates the difference between the amDateAdd() and amDateAddLogical() functions. A duration of 30 days will be added to the date 1/1/1999 (January 1, 1999) using each of these functions.

AmDateAdd adds a real duration, 30 days in this case:

```
RetVal=AmDateAdd("1999/01/01", 2592000)
```

The function returns the value:

```
1999/01/31
```

AmDateAddLogical adds a logical duration, in this case 30 days (=1 month):

```
RetVal=AmDateAddLogical("1999/01/01", 2592000)
```

The function returns the value:

```
1999/02/01
```

AmDateAddLogical()

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

API syntax

```
long AmDateAddLogical(long tmStart, long tsDuration);
```

Internal Basic syntax

```
Function AmDateAddLogical(tmStart As Date, tsDuration As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example illustrates the difference between the amDateAdd() and amDateAddLogical() functions. A duration of 30 days will be added to the date 1/1/1999 (January 1, 1999) using each of these functions.

AmDateAdd adds a real duration, 30 days in this case:

```
RetVal=AmDateAdd("1999/01/01", 2592000)
```

The function returns the value:

```
1999/01/31
```

AmDateAddLogical adds a logical duration, in this case 30 days (=1 month):

```
RetVal=AmDateAddLogical("1999/01/01", 2592000)
```

The function returns the value:

```
1999/02/01
```

AmDateDiff()

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

API syntax

```
long AmDateDiff(long tmEnd, long tmStart);
```

Internal Basic syntax

```
Function AmDateDiff(tmEnd As Date, tmStart As Date) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

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

AmDateDiffEx()

This function calculates the difference between two dates in seconds.

API syntax

```
long AmDateDiffEx(const struct tm *ptmEnd,const struct tm *ptmStart);
```

Field of application

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *ptmEnd*:This parameter contains a datetime value of the end.
- *ptmStart*: This parameter contains a datetime value of the start.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbExecAql()

This function enables you to execute an AQL query on the database.

API syntax

```
long AmDbExecAql(handle hApiCnxBase, char *strAqlStatement);
```

Internal Basic syntax

```
Function AmDbExecAql(strAqlStatement As String) As Long
```

Field of application

Version: 4.1.0

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strAqlStatement*: This parameter contains the AQL query to execute.

Output parameters

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

AmDbGetDate()

This function returns the result, in date format, of the AQL query. If the query does not return a result, the value 0 is returned without triggering an error.

API syntax

```
long AmDbGetDate(handle hApiCnxBase, char *strQuery);
```

Internal Basic syntax

```
Function AmDbGetDate(strQuery As String) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the full AQL query whose result you want to recover.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbGetDouble()

This function returns the result (as a double-precision number), of the AQL query. If the query does not return a result, the value 0 is returned without triggering an error.

API syntax

```
double AmDbGetDouble(handle hApiCnxBase, char *strQuery);
```

Internal Basic syntax

```
Function AmDbGetDouble(strQuery As String) As Double
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the full AQL query whose result you want to recover.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbGetFloat()

This function gets the query's result as float.

API syntax

```
float AmDbGetFloat(const char *strQuery);
```

Internal Basic syntax

```
Function AmDbGetFloat(strQuery As String) As Single
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.

Output parameters

- The result of query as float.

AmDbGetLimitedList()

This function returns the result of an AQL query in a numbered list. Unlike the `AmDbGetList` and `AmDbGetListEx` functions, this function is used to define the maximum number of elements selected by the AQL query and indicates what should to do if data is truncated.

API syntax

```
long AmDbGetLimitedList(handle hApiCnxBase, char *strQuery, char
*pstrResult, long lResult, char *strColSep, char *strLineSep, char
*strIdSep, long lMaxSize, long lErrorType);
```

Internal Basic syntax

```
Function AmDbGetLimitedList(strQuery As String, strColSep As String,
strLineSep As String, strIdSep As String, lMaxSize As Long, lErrorType As
Long) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the AQL query that is to be executed.
- *strColSep*: This parameter contains the character used as the column separator in the result returned by the function.
- *strLineSep*: This parameter contains the character used as the line separator in the result returned by the function.
- *strIdSep* : This parameter contains the character used as the identifier separator in the result returned by the function.
- *lMaxSize*: This parameter contains the maximum number of elements returned by the AQL query before truncation occurs.

- *lErrorType*: This parameter specifies what the function should do if data is truncated:
  - If the parameter is set to 1, the function returns an error message.
  - If the parameter is set to 2, the function returns a warning message.
  - If the parameter is set to 4, the function returns an informational message.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbGetList()

This function returns, as a list, the result of an AQL query. The number of elements selected by the AQL query is limited to 99.

API syntax

```
long AmDbGetList(handle hApiCnxBase, char *strQuery, char *pstrResult,
long lResult, char *strColSep, char *strLineSep, char *strIdSep);
```

Internal Basic syntax

```
Function AmDbGetList(strQuery As String, strColSep As String, strLineSep
As String, strIdSep As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the AQL query you want to execute.
- *strColSep*: This parameter contains the character used as the column separator in the result returned by the function.
- *strLineSep*: This parameter contains the character used as the line separator in the result returned by the function.
- *strIdSep*: This parameter contains the character used as the identifier separator in the result returned by the function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbGetListEx()

This function returns, as a list, the result of an AQL query. Unlike the `AmDbGetList` function, this function is not limited in the number of elements selected by the AQL query.

API syntax

```
long AmDbGetListEx(handle hApiCnxBase, char *strQuery, char *pstrResult,
long lResult, char *strColSep, char *strLineSep, char *strIdSep);
```

Internal Basic syntax

```
Function AmDbGetListEx(strQuery As String, strColSep As String,
strLineSep As String, strIdSep As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the AQL query you want to execute.
- *strColSep* : This parameter contains the character used as the column separator in the result returned by the function.
- *strLineSep*: This parameter contains the character used as the line separator in the result returned by the function.
- *strIdSep*: This parameter contains the character used as the identifier separator in the result returned by the function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If data that is returned by the `AmDbGetList` function contains characters used as column, line or identifier separators, these characters must be escaped using the '\' character.

We recommend that you use the `UnEscapeSeparators` function to delete the escape characters from strings returned by `AmDbGetList`.

AmDbGetLong()

This function returns the result of an AQL query. If the query does not return a result, the value 0 is returned without triggering an error.

API syntax

```
long AmDbGetLong(handle hApiCnxBase, char *strQuery);
```

Internal Basic syntax

```
Function AmDbGetLong(strQuery As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the full AQL query whose result you want to recover.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example returns the identifier of a product supplier:

```
AmDbGetLong("SELECT lSuppId FROM amProdSupp WHERE lProdId="+Str([ProdId])+")
```

AmDbGetPk()

This function returns the primary key of a table according to the WHERE clause in an AQL query. If the query does not return a result, the value 0 is returned without triggering an error.

API syntax

```
long AmDbGetPk(handle hApiCnxBase, char *strTableName, char *strWhere);
```

Internal Basic syntax

```
Function AmDbGetPk(strTableName As String, strWhere As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: SQL name of the table whose primary key you want to recover.
- *strWhere*: WHERE clause in an AQL query.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDbGetString()

This function returns the result of an AQL query as a formatted string. The number of elements selected by the AQL query is limited to 99. It returns a maximum of 254 characters per field (if there are more, the string is truncated).

> Do not use this function to recover the value of a single string type field. This functionis similar to the AmDbGetList and AmDbGetListEx functions.

API syntax

```
long AmDbGetString(handle hApiCnxBase, char *strQuery, char *pstrResult,
long lResult, char *strColSep, char *strLineSep);
```

Internal Basic syntax

```
Function AmDbGetString(strQuery As String, strColSep As String,
strLineSep As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the AQL query you want to execute.
- *strColSep*: This parameter contains the character used as column separator in the final string.
- *strLineSep*: This parameter contains the character used as line separator in the final string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

In the API syntax, the lResult parameter must contain the expected size of the resulting value.

Example

```
Dim strList As String
strList = amDbGetList("Select Name, FullName from amEmplDept Where Name Like 'C%'",
"|", ",", "=")
```

returns the string:

```
Carpenter|/Taltek/I.S. Department/Carpenter\, Jerome\, DEMO-
M016/=23459,Chavez|/Taltek/I.S. Department/Chavez\, Philip\, DEMO-
M014/=23460,Chouraqui|/Taltek/Sales/Los Angeles Agency/Chouraqui\, Thomas\, DEMO-
M017/=23491,Cipriani|/Taltek/Sales/Los Angeles Agency/Cipriani\, Fred\, DEMO-
M018/=23492,Clech|/Taltek/Sales/Burbank Agency/Clech\, Richard\, DEMO-
M021/=23482,Colombo|/Taltek/Finance/Colombo\, Gerald\, DEMO-M022/=23441
```

The escape character \ is used before commas.

The same query with `amDbGetString()` does not add escape characters, which makes it inappropriate to fill a list. For example:

```
amDbGetString("Select FullName from amEmplDept Where Name Like 'C%'", "|", chr(10),
"")
```

displays:

```
/Taltek/I.S. Department/Carpenter, Jerome, DEMO-M016/
/Taltek/I.S. Department/Chavez, Philip, DEMO-M014/
/Taltek/Sales/Los Angeles Agency/Chouraqui, Thomas, DEMO-M017/
/Taltek/Sales/Los Angeles Agency/Cipriani, Fred, DEMO-M018/
/Taltek/Sales/Burbank Agency/Clech, Richard, DEMO-M021/
/Taltek/Finance/Colombo, Gerald, DEMO-M022/
```

AmDbGetStringEx()

This function returns, as a character string , the result of an AQL query. The difference with the `AmDbGetString` function is that this function is not limited in the number of elements selected by the AQL query.

> Do not use this function to recover the value of a single string type field. This functionis similar to the AmDbGetList and AmDbGetListEx functions.

API syntax

```
long AmDbGetStringEx(handle hApiCnxBase, char *strQuery, char
*pstrResult, long lResult, char *strColSep, char *strLineSep);
```

Internal Basic syntax

```
Function AmDbGetStringEx(strQuery As String, strColSep As String,
strLineSep As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strQuery*: This parameter contains the AQL query you want to execute.
- *strColSep*: This parameter contains the character used as column separator in the final string.
- *strLineSep*: This parameter contains the character used as line separator in the final string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDeadLine()

This function calculates a deadline according to a calendar, a start date and a number of working seconds elapsed.

API syntax

```
long AmDeadLine(handle hApiCnxBase, char *strCalendarSqlName, long tmStart, long tsDuration);
```

Internal Basic syntax

```
Function AmDeadLine(strCalendarSqlName As String, tmStart As Date, tsDuration As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strCalendarSqlName*: This parameter contains the SQL name of the calendar of working periods used as a basis for calculating the deadline.

- *tmStart*: This parameter contains the start date of the period.
- *tsDuration*: This parameter contains the number of working seconds since the start date of the period.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example calculates the deadline according to the calendar whose SQL name is "Calendar_Paris", from a period start date set to 01/09/1998 at 08:00 and for a number of seconds equal to 450,000.

```
AmDeadLine("Calendar_Paris", "1998/09/01 08:00:00", 450000)
```

This example returns the deadline, that is, 22/09/1998 at 18:00.

AmDecrementLogLevel()

This function enables you to go up one level in the hierarchy of a log window in the final page of a wizard.

Internal Basic syntax

```
Function AmDecrementLogLevel() As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDefAssignee()

This function searches for the ID number of the default ticket supervisor for a given employee group.

API syntax

```
long AmDefAssignee(handle hApiCnxBase, long lGroupId);
```

Internal Basic syntax

```
Function AmDefAssignee(lGroupId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lGroupId*: This parameter contains the ID number of an employee group.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following generic example returns the identifier of the default supervisor for an employee group:

```
AmDefAssignee([lGroupId])
```

You can directly enter the numeric value of the identifier, as in the following example:

```
AmDefAssignee(24)
```

AmDefaultCurrency()

Returns the default currency used in Asset Manager.

API syntax

```
long AmDefaultCurrency(handle hApiCnxBase, char *return, long lreturn);
```

Internal Basic syntax

```
Function AmDefaultCurrency() As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmDefEscalationScheme()

This function searches for the default escalation scheme according to the location and severity of the helpdesk ticket.

API syntax

```
long AmDefEscalationScheme(handle hApiCnxBase, char *strLocFullName, long lSeverityLvl);
```

Internal Basic syntax

```
Function AmDefEscalationScheme(strLocFullName As String, lSeverityLvl As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strLocFullName*: This parameter contains the full name of the location.
- *lSeverityLvl*: This parameter contains the value of the severity.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following generic example returns the identifier of the default escalation scheme according to the location and the severity:

```
AmDefEscalationScheme([Asset.Location.FullName], [Severity.lSeverityLvl])
```

You can directly enter the value of the parameters, as in the following example:

```
AmDefEscalationScheme ( "/Location/", 24)
```

AmDefGroup()

This function returns the ID number of the default helpdesk group according to the type of problem, the location, and the maintenance contract.

API syntax

```
long AmDefGroup(handle hApiCnxBase, long lProblemClassId, char
*strLocFullName, long lAssetMainCntId);
```

Internal Basic syntax

```
Function AmDefGroup(lProblemClassId As Long, strLocFullName As String,
lAssetMainCntId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lProblemClassId*: This parameter contains the ID number for a problem type.
- *strLocFullName*: This parameter contains the full name of a location.
- *lAssetMainCntId*: This parameter contains the ID number of a maintenance contract.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The method used to define the default helpdesk group is the following:

1. The function searches the helpdesk groups associated with the problem type of the ticket.
2. From these groups, the function searches the helpdesk groups associated with the "nearest" location to the asset: direct location, else parent location, and so on until the root location.
3. If no group is found, and if the DBMS supports double outer joins, the function searches groups that aren't associated with a location.
   For the list of DBMSs that support double external joints, refer to the **Helpdesk** guide, chapter **References (Helpdesk)**, section **DBMSs supporting double outer-joins**.
4. If the DBMS supports double outer joins, the function selects, from the groups found previously, the helpdesk group linked to maintenance contracts covering the asset.
5. If no group is found, the function repeats steps 1, 2, 3 and 4 starting from the problem type a level up in the hierarchy of problems until it reaches the root of the problem-type tree.

Example

The following generic example calculates the ID number of the default helpdesk group according to three parameters: the type of problem, the location, and the maintenance contract.

```
AmDefGroup([ProblemClass.lPbClassId],[Asset.Location.FullName],
[Asset.lMaintCntrId])
```

You can directly enter the numeric value of the parameters using the ID numbers, as shown in the following example:

```
AmDefGroup(0, [Asset.Location.FullName], 0)
```

AmDeleteLink()

This function deletes a links of a record.

API syntax

```
long AmDeleteLink(handle hApiRecord, char *strLinkName, handle
hApiRecDest);
```

Internal Basic syntax

```
Function AmDeleteLink(hApiRecord As Long, strLinkName As String,
hApiRecDest As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record containing the link to be deleted.

- *strLinkName*: This parameter contains the SQL name of the link to be deleted.
- *hApiRecDest*: This parameter contains a handle of the target record of the link to be deleted.

Output parameters

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

AmDeleteRecord()

This function deletes a record in the database.

API syntax

```
long AmDeleteRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmDeleteRecord(hApiRecord As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains a handle of the record you want to delete.

Output parameters

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

AmDisconnectTrace()

The AmDisconnectTrace API disconnects the trace between a user node (lEndId) and host node (lStartId) in the cable link table. If either node is at the end of a trace, it will be deleted from the cable link table. It also creates trace history and trace operations entries based on the disconnect.

API syntax

```
long AmDisconnectTrace(handle hApiCnxBase, long lStartId, long lEndId,
char *strComment, long lCabTraceOutId);
```

Internal Basic syntax

```
Function AmDisconnectTrace(lStartId As Long, lEndId As Long, strComment
As String, lCabTraceOutId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓

	Available
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lStartId*: This parameter defines the host connection ID that will be disconnected.
- *lEndId*: This parameter defines the user connection ID that will be disconnected.
- *strComment*: This parameter is the string operation to show new connects and disconnects.
- *lCabTraceOutId*: This parameter is the cable trace output ID.

Output parameters

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

AmDuplicateRecord()

This function enables you to duplicate a record.

API syntax

```
long AmDuplicateRecord(handle hApiRecord, long bInsert);
```

Internal Basic syntax

```
Function AmDuplicateRecord(hApiRecord As Long, bInsert As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record to duplicate.
- *bInsert*: This parameter enables you to specify whether you want to insert the duplicated record immediately (=1) or not (=0).

Output parameters

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

AmEndOfNthBusinessDay()

Gives the last business hour of the nth day (identified by the integer *lDayCount*) from a given date according to a calendar.

API syntax

```
long AmEndOfNthBusinessDay(handle hApiCnxBase, char *strCalendarSqlName,
long tmStart, long lDayCount);
```

Internal Basic syntax

```
Function AmEndOfNthBusinessDay(strCalendarSqlName As String, tmStart As
Date, lDayCount As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strCalendarSqlName*: Name of the calendar used for the calculation.
- *tmStart*: Start date for the calculation.
- *lDayCount*: Number of full business days to add to dStart for the calculation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmEnumValList()

This function returns a string containing all the values of a custom itemized list. The different values are sorted alphabetically and are delimited by the separator indicated in the *strLineSep* parameter.

If an itemized list value contains the character used as the separator or a "\", the "\" prefix is used.

API syntax

```
long AmEnumValList(handle hApiCnxBase, char *strEnumName, char
*pstrValList, long lValList, long bNoCase, char *strLineSep);
```

Internal Basic syntax

```
Function AmEnumValList(strEnumName As String, bNoCase As Long, strLineSep
As String) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strEnumName*: This parameter contains the SQL name of the itemized list for which you want to recover the values.
- *bNoCase*: This parameter enables you to specify whether the sort is case sensitive (=1) or not (=0).
- *strLineSep*: This parameter contains the character used to delimit the itemized-list values.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmESDAddComputers()

Internal Basic syntax

```
Function AmESDAddComputers(lESDTaskId As Long, selTarget As String,
lplIgnoredCount As Long) As Long
```

Field of application

Version: 5.0.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmESDCreateTask()

Internal Basic syntax

```
Function AmESDCreateTask(strDescription As String, lESDDelivMethodId As
Long, lESDPackageId As Long, dttimeStart As Date, selTarget As String,
bStart As Long, lplIgnoredCount As Long) As Long
```

Field of application

Version: 5.0.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmEvalScript()

This function enables you to evaluate a script by its name from the current context. This function has two uses:

- Evaluate a system script (Default value, Mandatory, and so on)
- Call a function from a script library

Internal Basic syntax

```
Function AmEvalScript(strScriptName As String, strObject As String,
strPath As String, ...) As Variant
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	

Input parameters

- *strScriptName*: This parameter contains the name of the script to evaluate. In the first case, it is the name of the system script (DefVal, and so on). In the second case, it is the name of a script library.
- *strObject*: This parameter contains the object concerned by the script. It can be the SQL name of a field or the name of a function from the library.
- *strPath*: This optional parameter enables you to specify a path (link.link.link...) to shift the context of evaluation of a script. This parameter does not work in the second case.
- *...*: When calling a function from a script library, enables you to pass parameters to the function called.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The following is a list of usable system script names:

- For a table: IsValid, IsRelevant
- For a field: DefVal, Mandatory, Historized, ReadOnly, Irrelevant
- For a link: Historized, Filter, Irrelevant
- For a feature: DefVal, Mandatory, Available, Historized

AmExecTransition()

This function triggers a valid transition from the current page.

Internal Basic syntax

```
Function AmExecTransition(strTransName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strTransName*: This parameter contains the name of the transition as defined in the wizard script. An error is returned if the transition is not found. The function does not work (and does not return an error) if the transition is not valid.

Output parameters

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

AmExecuteActionById()

This function executes an action as identified by its identifier.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

API syntax

```
long AmExecuteActionById(handle hApiCnxBase, long lActionId, char
*strTableName, long lRecordId);
```

Internal Basic syntax

```
Function AmExecuteActionById(lActionId As Long, strTableName As String,
lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lActionId*: This parameter contains the identifier of the action to be executed.
- *strTableName*: In the case of a contextual action, this parameter contains the SQL name of the table on which the action is executed. If this parameter is omitted, in the case of a contextual action, the function will fail. For non-contextual actions, this parameter is not interpreted and therefore optional.
- *lRecordId*: This parameter contains the identifier of a possible record concerned by the action. For non-contextual actions, this parameter is not interpreted and therefore optional.

Output parameters

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

AmExecuteActionByName()

This function executes an action as identified by its SQL name.

> **Note:** If the API is triggered on Asset Manager web, it is executed on the web service server.

API syntax

```
long AmExecuteActionByName(handle hApiCnxBase, char *strSqlName, char
*strTableName, long lRecordId);
```

Internal Basic syntax

```
Function AmExecuteActionByName(strSqlName As String, strTableName As
```

```
String, lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strSqlName*: This parameter contains the SQL name of the action to be executed.
- *strTableName*: In the case of a contextual action, this parameter contains the SQL name of the table on which the action is executed. If this parameter is omitted, in the case of a contextual action, the function will fail. For non-contextual actions, this parameter is not interpreted and therefore optional.
- *lRecordId*: This parameter contains the identifier of a possible record concerned by the action. For non-contextual actions, this parameter is not interpreted and therefore optional.

Output parameters

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

AmExportDocument()

This function enables you to export a document attached to a record.

API syntax

```
long AmExportDocument(handle hApiCnxBase, long lDocId, char
*strFileName);
```

Internal Basic syntax

```
Function AmExportDocument(lDocId As Long, strFileName As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lDocId*: This parameter contains the identifier of the document to export.
- *strFileName*: This parameter contains the name of the document to export, as it is stored in the FileName field of the Documents table.

Output parameters

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

AmExportReport()

This function enables you to export a SAP Crystal Report from the database.

Internal Basic syntax

```
Function AmExportReport(lReportId As Long, strFileName As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lReportId*: This parameter contains the identifier of the SAP Crystal Reports record to be exported.
- *strFileName*: This parameter contains the full path of the file to which the export is made.

Output parameters

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

AmFindCable()

The AmFindCable API finds the next available cable that runs between a given user location (lUserId) and host location (lHostId). The cable must be of the specified cable type (strCabType) and cable role (strCableRole). The cable must also have a status of "Available". The cables are sorted in ascending order by cable ID and only cables greater than the previous cable ID (lPrevCabId) are selected.

API syntax

```
long AmFindCable(handle hApiCnxBase, long lPrevCableId, char *strCabType,
long lUserId, long lHostId, char *strCableRole);
```

Internal Basic syntax

```
Function AmFindCable(lPrevCableId As Long, strCabType As String, lUserId
As Long, lHostId As Long, strCableRole As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lPrevCableId*: This parameter is the ID of the previous cable.
- *strCabType*: This parameter defines the cable type for searching.
- *lUserId*: This parameter defines the user location ID.
- *lHostId*: This parameter defines the host location ID.
- *strCableRole*: This parameter is the cable role to locate.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmFindDevice()

The AmFindDevice API finds a device of a given type (strDevType) in a given location (lLocId). The devices are sorted in ascending order by device ID and only devices greater than the previous device ID (lPrevDeviceId) are selected.

API syntax

```
long AmFindDevice(handle hApiCnxBase, long lPrevDeviceId, char
*strDeviceType, long lLocationId);
```

Internal Basic syntax

```
Function AmFindDevice(lPrevDeviceId As Long, strDeviceType As String,
lLocationId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`lPrevDeviceId`*: This parameter defines the previous device ID searched. The value of 0 is used

to start a search.

- *strDeviceType*: This parameter defines the device type to locate.
- *lLocationId*: This parameter is the location ID to search.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmFindRootLink()

This function enables you to recover the root link of a trace.

API syntax

```
long AmFindRootLink(handle hApiCnxBase, long lLinkId);
```

Internal Basic syntax

```
Function AmFindRootLink(lLinkId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lLinkId*: This parameter contains the identifier of the link concerned by the operation.

Output parameters

The function returns the identifier of the root link.

AmFindTermDevice()

The AmFindTermDevice API finds the next available device in a given termination field (lTermField) for a given cable role (strCableRole). The devices are sorted in ascending order by sequence number and only assets greater than the previous sequence number (strPrevTermSeq) are selected. Also, for pin-based devices (bPinBased = 1), we check the total number of pins needed (iPinPortCount) against the total number of pins remaining on the device. For port-based devices (bPinBased = 0) we check to make sure there is at least one port remaining on the device and that the remaining port has the host or user side available by the checking the flag (bCheckAvail = 0 - user device, bCheckAvail = 1 - host device).

API syntax

```
long AmFindTermDevice(handle hApiCnxBase, long iPrevTermSeq, long
lTermFieldId, char *strCableRole, long bPinBased, long iPinPortCount,
long bCheckAvail);
```

Internal Basic syntax

```
Function AmFindTermDevice(iPrevTermSeq As Long, lTermFieldId As Long,
strCableRole As String, bPinBased As Long, iPinPortCount As Long,
bCheckAvail As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iPrevTermSeq*: This parameter is the previous termination field's sequence searched. The value of 0 is used to start a search.
- *lTermFieldId*: This parameter is the termination field ID.
- *strCableRole*: This parameter is the cable role to locate.
- *bPinBased*: This parameter determines whether the device is pin-based or port-based.
- *iPinPortCount*: For pin-based devices, this parameter is the total number of pins needed to create a virtual port. For port-based devices, this parameter is 1 since this API is called per port needed.
- *bCheckAvail*: This parameter is used to determine what side of the port needs to be available.
  - 0=user device, check host available
  - 1=host device, check user available

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmFindTermField()

The AmFindTermField API finds a termination field that provides the given duty (lDutyId) from the given location (lLocId). It will continue to find additional termination fields in a given location for a given duty if lTermFieldId is greater than 0.

API syntax

```
long AmFindTermField(handle hApiCnxBase, long lDutyId, long lLocationId,
long lPrevTermFieldId);
```

Internal Basic syntax

```
Function AmFindTermField(lDutyId As Long, lLocationId As Long,
lPrevTermFieldId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lDutyId*: This parameter defines the duty to locate.

- *lLocationId*: This parameter is the location ID to search.
- *lPrevTermFieldId*: This parameter is the termination field ID.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmFlushTransaction()

This function purges the task lists of the agents (like after a database Commit operation).

API syntax

```
long AmFlushTransaction(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmFlushTransaction() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmFormatCurrency()

This function displays a monetary value in a given currency. The standard symbol of the currency is also displayed.

API syntax

```
long AmFormatCurrency(double dAmount, char *strCurrency, char
*pstrDisplay, long lDisplay);
```

Internal Basic syntax

```
Function AmFormatCurrency(dAmount As Double, strCurrency As String) As
String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *dAmount*: This parameter contains the monetary value to be displayed.
- *strCurrency*: This parameter contains the currency used for the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal=amFormatCurrency(500, "USD")
```

This example displays:

```
US$500.00
```

AmFormatLong()

This function remplaces a token in a character string with the value contained in a Long type variable.

API syntax

```
long AmFormatLong(handle hApiCnxBase, long lNumber, char *strFormat, char
*pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmFormatLong(lNumber As Long, strFormat As String) As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lNumber*: This parameter contains the Long to be inserted in the character string contained in the *strFormat* parameter.
- *strFormat*: This parameter contains the character string to process. All "%d" type tokens are remplaced with the value contained in the *lNumber* parameter.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGeneratePlanningData()

This function enables you to generate a graphical planner viewer.

Internal Basic syntax

```
Function AmGeneratePlanningData(strTableSqlName As String, strProperties
As String, strIds As String) As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableSqlName*: This parameter contains the SQL name of the table containing the data from which the schedule is generated.
- *strProperties*: This parameter contains the properties of the created schedule.

  > For more information on the syntax of these properties, refer to the Administration Guide, References: parameter syntax of the planner viewer pages.

- *strIds*: This parameter contains the list of identifiers (separated by commas) of the records whose data is used to create the schedule.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGenSqlName()

This function generates a valid SQL name from a classic string. Spaces are replaced by underscores ("_"). This function is especially useful for defining the default value of a SQL name for a feature based on its name.

API syntax

```
long AmGenSqlName(char *return, long lreturn, char *strText);
```

Internal Basic syntax

```
Function AmGenSqlName(strText As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strText*: Character string used to generate the SQL name.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example defines the default value of the SQL name of an object named "Label" in the Asset Manager database:

```
RetVal=AmGenSQLName([Label])
```

AmGetCatRef()

This function searches for a valid non-catalog reference for a given model (respecting the validity dates). The following rules are respected:

- `CatProduct.lModelId=lModelId`
- `CatProduct.lParentId=0`

The function returns a reference not created on the fly as its priority. If no references are found and the parameter *`bCreate`* is set to "1", a new non-catalog reference and a product are created (pointing to the model).

API syntax

```
long AmGetCatRef(handle hApiCnxBase, long lModelId, long bCreate);
```

Internal Basic syntax

```
Function AmGetCatRef(lModelId As Long, bCreate As Long) As Long
```

Field of application

Version: 4.1.0

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lModelId*: This parameter contains the ID of the model concerned by the operation.
- *bCreate*: This parameter enables you to specify if a non-catalog reference should be created in the case where the search returns no results.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The function does not require you to specify a supplier. This is because the searchis performed on non-catalog references regardless of the supplier.

AmGetCatRefFromCatProduct()

This function is identical to the function `amGetCatRef`, except that the search is performed for a given product.

API syntax

```
long AmGetCatRefFromCatProduct(handle hApiCnxBase, long lCatProductId,
```

```
long bCreate);
```

Internal Basic syntax

```
Function AmGetCatRefFromCatProduct(lCatProductId As Long, bCreate As
Long) As Long
```

Field of application

Version: 4.1.0

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCatProductId*: This parameter contains the ID of the product concerned by the operation.
- *bCreate*: This parameter enables you to specify if a non-catalog reference should be created in the case where the search returns no results.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetComputeString()

This function returns the description string of a given record according to a template.

API syntax

```
long AmGetComputeString(handle hApiCnxBase, char *strTableName, long lRecordId, char *strTemplate, char *pstrComputeString, long lComputeString);
```

Internal Basic syntax

```
Function AmGetComputeString(strTableName As String, lRecordId As Long, strTemplate As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: This parameter contains the SQL name of the table of the record for which you want to recover the description string.
- *lRecordId*: This parameter contains the identifier of the record within the table.
- *strTemplate*: This parameter contains, in the form of a character string, the template used for the description string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal = amGetComputeString("amEmplDept", [lEmplDeptId], "[Name], [FirstName]")
```

AmGetConnection()

The AmGetConnection API opens a new connection or retrieve one from AM connection pool.

API syntax

```
handle AmGetConnectionU(char *strDataSource, char *strSudoerUser, char
*strSudoerPwd, char *strUser);
```

> **Note: strDataSource** must be a valid data source (data sources are listed in the Asset Manager connection box).

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	

	Available
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- **strDataSource**: Name of the data source.
- **strSudoerUser**: Impersonated user name.
- **strSudoerPwd**: Password of the Impersonated user.
- **strUser**: User name for the connection.

Output parameters

- Handle of AM session.

AmGetCurrentNTDomain()

This function returns the name of the NT domain of the current login.

API syntax

```
long AmGetCurrentNTDomain(char *pstrDomain, long lDomain);
```

Internal Basic syntax

```
Function AmGetCurrentNTDomain() As String
```

Field of application

Version: 4.00

	Available
AssetManager API	

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal = amGetCurrentNTDomain()
```

AmGetCurrentNTUser()

This function enables you to get the login of the user connected to Windows (NT or 2000).

API syntax

```
long AmGetCurrentNTUser(char *pstrUser, long lUser);
```

Internal Basic syntax

```
Function AmGetCurrentNTUser() As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFeat()

This function creates a feature object from a handle to a table name and returns the handle of the created feature object.

API syntax

```
handle AmGetFeat(handle hApiTable, long lPos);
```

Internal Basic syntax

```
Function AmGetFeat(hApiTable As Long, lPos As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a handle of a table.
- *`lPos`*: This parameter contains the position of the feature in the table.

AmGetFeatCount()

This function returns the number of features of the table specified in the *`hApiTable`* parameter.

API syntax

```
long AmGetFeatCount(handle hApiTable);
```

Internal Basic syntax

```
Function AmGetFeatCount(hApiTable As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiTable*: This parameter contains a handle of a table.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetField()

This function creates a field object from the handle of a query, a record or a table and returns the handle of the field object created.

API syntax

```
handle AmGetField(handle hApiObject, long lPos);
```

Internal Basic syntax

```
Function AmGetField(hApiObject As Long, lPos As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query, record, or table.
- *lPos*: This parameter contains the position of the field (its index) within the object.

AmGetFieldCount()

This function returns the number of fields contained in the current object.

API syntax

```
long AmGetFieldCount(handle hApiObject);
```

Internal Basic syntax

```
Function AmGetFieldCount(hApiObject As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a valid record, query or table.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldDateOnlyValue()

This function returns the value of a field contained in the current object. This value is returned in the "Date" format (from an external tool, it is a Long). Unlike the `AmGetFieldDateValue` function, only the Date part is returned, the Time part is omitted.

API syntax

```
long AmGetFieldDateOnlyValue(handle hApiObject, long lFieldPos);
```

Internal Basic syntax

```
Function AmGetFieldDateOnlyValue(hApiObject As Long, lFieldPos As Long)
As Date
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query or record.
- *lFieldPos*: This parameter contains the number of the field within the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldDateValue()

This function returns the value of a field contained in the current object. This value is returned in "Date" format (from external tools, it is a Long).

API syntax

```
long AmGetFieldDateValue(handle hApiObject, long lFieldPos);
```

Internal Basic syntax

```
Function AmGetFieldDateValue(hApiObject As Long, lFieldPos As Long) As
Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *hApiObject*: This parameter contains a handle of a query or record.
- *lFieldPos*: This parameter contains the number of the field inside the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldDescription()

This function returns, as a character string ("String" format), the long description of a field identified by a handle.

API syntax

```
long AmGetFieldDescription(handle hApiField, char *pstrBuffer, long
lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldDescription(hApiField As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *hApiField*: This parameter contains a valid handle of the field whose long description you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldDoubleValue()

This function returns the value of a field contained in the current object. This value is returned in "Double" format.

API syntax

```
double AmGetFieldDoubleValue(handle hApiObject, long lFieldPos);
```

Internal Basic syntax

```
Function AmGetFieldDoubleValue(hApiObject As Long, lFieldPos As Long) As
Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query or record.

- *lFieldPos*: This parameter contains the number of the field inside the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldFormat()

This function is useful when the value of the "UserType" of the field concerned (cf. "database.txt" file) is:

- System itemized list
- Itemized list
- Time span
- Table or field name

The function returns the format of the "UserType", that is:

UserType	Format returned by the function
System itemized list	List of system-itemized list entries.
Itemized list	Name of the itemized list associated to the field.
Time span	Display format.
Table or field name	SQL name of the field that stores the SQL name of the table.

API syntax

```
long AmGetFieldFormat(handle hApiField, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldFormat(hApiField As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle of the field whose "UserType" you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldFormatFromName()

This function returns the "UserType" format of a field, from its name.

API syntax

```
long AmGetFieldFormatFromName(handle hApiCnxBase, char *strTableName,
char *strFieldName, char *pFieldFormat, long lpFieldFormat);
```

Internal Basic syntax

```
Function AmGetFieldFormatFromName(strTableName As String, strFieldName As
String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: This parameter contains the SQL name of the table containing the field concerned by the operation.
- *strFieldName*: This parameter contains the SQL name of the field.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldFromName()

This function creates a field object based on its name and returns the handle of the field object created.

API syntax

```
handle AmGetFieldFromName(handle hApiObject, char *strName);
```

Internal Basic syntax

```
Function AmGetFieldFromName(hApiObject As Long, strName As String) As
Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query, record, or table.
- *strName*: This parameter contains the field name.

AmGetFieldLabel()

This function returns, as a character string ("String" format), the label of a field identified by a handle.

API syntax

```
long AmGetFieldLabel(handle hApiField, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldLabel(hApiField As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a valid handle of the field whose label you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldLabelFromName()

This function returns the label of a field from its SQL name.

API syntax

```
long AmGetFieldLabelFromName(handle hApiCnxBase, char *strTableName, char *strFieldName, char *pFieldLabel, long lpFieldLabel);
```

Internal Basic syntax

```
Function AmGetFieldLabelFromName(strTableName As String, strFieldName As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: This parameter contains the SQL name of the table containing the field concerned by the operation.
- *strFieldName*: This parameter contains the SQL name of the field.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldLongValue()

This function returns the value of a field contained in the current object.

API syntax

```
long AmGetFieldLongValue(handle hApiObject, long lFieldPos);
```

Internal Basic syntax

```
Function AmGetFieldLongValue(hApiObject As Long, lFieldPos As Long) As
Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query or record.
- *lFieldPos*: This parameter contains the number of the field inside the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If you use this function to recover the value of a field of date, time or date+time type,the long integer returned by the function represents the number of seconds since01/01/1970 at 00:00:00.

AmGetFieldName()

This function returns the name of a field contained in the current object.

API syntax

```
long AmGetFieldName(handle hApiObject, long lFieldPos, char *pstrBuffer,
long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldName(hApiObject As Long, lFieldPos As Long) As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query, record, or table.
- *lFieldPos*: This parameter contains the number of the field within the current object. E.g., the value "0" indicates the first field.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldRights()

This function returns the user rights for a field in the current object. These rights are returned as a character string containing three characters, which specify the read/insert/update rights:

- "r": indicates the right to read data.
- "i": indicates the right to insert data.
- "u": indicates the right to update data.

For example, for a read-only field, the function returns the value "r ".

API syntax

```
long AmGetFieldRights(handle hApiObject, long lFieldPos, char
*pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldRights(hApiObject As Long, lFieldPos As Long) As
String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query, record, or table.
- *lFieldPos*: This parameter contains the number of the field within the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldSize()

This function returns the size of a field.

API syntax

```
long AmGetFieldSize(handle hApiField);
```

Internal Basic syntax

```
Function AmGetFieldSize(hApiField As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a handle of the field whose size you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldSqlName()

This function returns, as a character string ("String" format), the SQL name of a field identified by a handle.

API syntax

```
long AmGetFieldSqlName(handle hApiField, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldSqlName(hApiField As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a valid handle of the field whose SQL name you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldStrValue()

This function returns the value of a field contained in the current object. This value is returned in string format.

Warning: When this function is used via the Asset Manager APIs, it expects two extra parameters *pszBuffer* and *lBuffer*, which define a string used as a buffer to store the recovered string and the size of this buffer respectively. The *pszBuffer* string must be formatted (filled with characters) and be of the size defined by *lBuffer*. The following portion of code is incorrect, the string used as a buffer is not sized:

```
Dim strBuffer as String
Dim lRec as Long
Dim lBuffer as Long
lBuffer=20
lRec=AmGetFieldStrValue(1, 0, strBuffer, lBuffer)
```

Here is the corrected portion of code:

```
Dim strBuffer as String
Dim lRec as Long
Dim lBuffer as Long
strBuffer=String(21, " ") ' The buffer is set to 21 characters (" ")
lBuffer=20
lRec=AmGetFieldStrValue(1, 0, strBuffer, lBuffer)
```

When you format a buffer string using the "String" function, do not use "0" as a padding character. Size the buffer before calling the `AmGetFieldStrValue` function, particularly if this function is in a loop and always uses the same string as a buffer.

API syntax

```
long AmGetFieldStrValue(handle hApiObject, long lFieldPos, char
*pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetFieldStrValue(hApiObject As Long, lFieldPos As Long) As
String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiObject`*: This parameter contains a handle of a query or record.
- *`lFieldPos`*: This parameter contains the number of the field inside the current object.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetFieldType()

This function returns the type of a field.

API syntax

```
long AmGetFieldType(handle hApiField);
```

Internal Basic syntax

```
Function AmGetFieldType(hApiField As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a handle of the field whose type you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The following table lists the values returned by the `AmGetFieldType` function for each type of field:

Values returned	Corresponding field type
0	Undefined
1	Byte
2	Short
3	Long
4	Float
5	Double
6	String
7	Time stamp
8	Bin
9	Blob
10	Date
11	Time
12	Memo

AmGetFieldUserType()

This function returns the "UserType" of a field (cf. **database.txt** file) identified by a handle, in the form of a long integer. For a field, the valid return values are summarized below:

Stored value	Plain-text value
0	Default
1	Number

Stored value	Plain-text value
2	Yes/ No
3	Money
4	Date
5	Date+Time
7	System itemized list
8	Custom itemized list
10	Percentage
11	Time span
12	Table or field SQL name

For a link, the valid return values are summarized below:

Stored value	Plain-text value
0	Normal
1	Comment
2	Image
3	History
4	Feature value

Up until version 4.0.0, the function always returned 0 for a link. From AssetCenter version 4.1.0 onwards, the function returns one of the following values for a link:

- 0: Normal
- 1: Comments
- 2: Image
- 3: History
- 5: Script

API syntax

```
long AmGetFieldUserType(handle hApiField);
```

Internal Basic syntax

```
Function AmGetFieldUserType(hApiField As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle of the field whose "UserType" you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetForeignKey()

Recovers the handle of the foreign key of a link, itself identified by its handle.

API syntax

```
handle AmGetForeignKey(handle hApiField);
```

Internal Basic syntax

```
Function AmGetForeignKey(hApiField As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: Handle of the link concerned by the operation.

AmGetIndex()

This function creates an index object from a handle of a query, record, or a table and returns the handle of the index object created.

API syntax

```
handle AmGetIndex(handle hApiTable, long lPos);
```

Internal Basic syntax

```
Function AmGetIndex(hApiTable As Long, lPos As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiTable*: This parameter contains a handle of a table.
- *lPos*: This parameter contains the position of the index in the table.

AmGetIndexCount()

This function returns the number of indexes contained in the table specified in the *hApiTable* parameter.

API syntax

```
long AmGetIndexCount(handle hApiTable);
```

Internal Basic syntax

```
Function AmGetIndexCount(hApiTable As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a handle of a table.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetIndexField()

This function returns a handle on a field identified by its position in the index (the lpos th field of the index).

API syntax

```
handle AmGetIndexField(handle hApiIndex, long lPos);
```

Internal Basic syntax

```
Function AmGetIndexField(hApiIndex As Long, lPos As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiIndex*: This parameter contains a valid handle on the index concerned by the operation.
- *lPos*: This parameter contains the position of the field in the index.

AmGetIndexFieldCount()

This function returns the number of fields making up an index.

API syntax

```
long AmGetIndexFieldCount(handle hApiIndex);
```

Internal Basic syntax

```
Function AmGetIndexFieldCount(hApiIndex As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiIndex*: This parameter contains a valid handle on the index concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetIndexFlags()

This function returns the parameters of an index.

API syntax

```
long AmGetIndexFlags(handle hApiIndex);
```

Internal Basic syntax

```
Function AmGetIndexFlags(hApiIndex As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiIndex*: This parameter contains a valid handle on the index concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The value returned by the function results from a logical combination (OR) of the following values:

- 1: The index authorized non unique records,
- 2: The index authorizes the null value,
- 4: The index is not case sensitive.

Thus, if the function returns 3, you can deduce that the index accepts non unique records and the null value (1 OR 2 = 3).

AmGetIndexName()

This function returns the name of an index.

API syntax

```
long AmGetIndexName(handle hApiIndex, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetIndexName(hApiIndex As Long) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiIndex*: This parameter contains a valid handle on the index whose name you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetLink()

This function creates a link object from the handle of a table and returns the handle of the link object created.

API syntax

```
handle AmGetLink(handle hApiTable, long lPos);
```

Internal Basic syntax

```
Function AmGetLink(hApiTable As Long, lPos As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a handle of a table.
- *`lPos`*: This parameter contains the position of the link (its index) inside the object.

AmGetLinkCardinality()

This function returns the cardinality of a link.

API syntax

```
long AmGetLinkCardinality(handle hApiField);
```

Internal Basic syntax

```
Function AmGetLinkCardinality(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a handle of the link whose cardinality you want to know.

Output parameters

- 1: The cardinality of the link is 1-1.
- 2: The cardinality of the link is 1-n.

AmGetLinkCount()

This function returns the number of links contained in the current table.

API syntax

```
long AmGetLinkCount(handle hApiTable);
```

Internal Basic syntax

```
Function AmGetLinkCount(hApiTable As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *hApiTable*: This parameter contains a handle of a valid table.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetLinkDstField()

This function returns the field (foreign key) to which the link defined by the *hApiField* parameter points.

API syntax

```
handle AmGetLinkDstField(handle hApiField);
```

Internal Basic syntax

```
Function AmGetLinkDstField(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a handle of the link concerned by the operation.

AmGetLinkFeatureValue()

Returns the value of a "Link" type feature.

API syntax

```
long AmGetLinkFeatureValue(handle hApiObject, long lFieldPos, long
lRecordId);
```

Internal Basic syntax

```
Function AmGetLinkFeatureValue(hApiObject As Long, lFieldPos As Long,
lRecordId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of a query or record.
- *lFieldPos*: This parameter contains the position of the field inside the current object.
- *lRecordId*: This parameter contains the identifier of the record whose feature value you want to recover.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim q as String
q = "Select fv_link, lEmplDeptId From amEmplDept Where lEmplDeptId = " &
[lEmplDeptId]
Dim hq as Long
hq = amQueryCreate()
Dim lErr as Long
lErr = amQueryGet(hq, q)
Dim lId as Long
lId = amGetFieldLongValue(hq, 1)
amMsgBox("str: " & amGetFieldStrValue(hq, 0))
amMsgBox("int: " &
amGetFieldLongValue(hq,0))
amMsgBox("lnk: " & amGetLinkFeatureValue(hq,0,lId))
```

AmGetLinkFromName()

This function creates a link object from a name and returns the handle of the object created.

API syntax

```
handle AmGetLinkFromName(handle hApiTable, char *strName);
```

Internal Basic syntax

```
Function AmGetLinkFromName(hApiTable As Long, strName As String) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a handle of a table.
- *`strName`*: This parameter contains the SQL name of the link.

AmGetLinkType()

This function returns the type of a link.

API syntax

```
long AmGetLinkType(handle hApiField);
```

Internal Basic syntax

```
Function AmGetLinkType(hApiField As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a handle of the link whose type you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetMainField()

This function creates a field object corresponding to the main field in a given table. It returns a handle of the field thus created.

API syntax

```
handle AmGetMainField(handle hApiTable);
```

Internal Basic syntax

```
Function AmGetMainField(hApiTable As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a handle of the table whose main field you want to know.

AmGetMemoField()

This function creates a field object that correspondes to a Memo-type field of a given table. It returns a handle on the field thus created.

API syntax

```
handle AmGetMemoField(handle hApiTable);
```

Internal Basic syntax

```
Function AmGetMemoField(hApiTable As Long) As Long
```

Field of application

Version: 4.1.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiTable*: This parameter contains a handle on the table whose Memo field is wanted.

AmGetNextAssetPin()

The AmGetNextAssetPin API finds the next available pin on a device (lAssetId). Its sequence number sorts the pins. Depending on the port direction (iPinPortDir), the available pins are sorted in ascending (iPinPortDir = 0) or descending (iPinPortDir = 1) order.

API syntax

```
long AmGetNextAssetPin(handle hApiCnxBase, long lAssetId, long
bPinPortDir, long iPrevPinSeq);
```

Internal Basic syntax

```
Function AmGetNextAssetPin(lAssetId As Long, bPinPortDir As Long,
iPrevPinSeq As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lAssetId*: This parameter is the device ID.
- *bPinPortDir*: This parameter is the direction to search.
  - 0=ascending
  - 1=descending
- *iPrevPinSeq*

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNextAssetPort()

The AmGetNextAssetPort API finds the next available port on a device (lAssetId) providing a given service (lDutyId) or no service at all. The status of the port must be "Available". Boolean flags are used to signify if the user side (bCheckUser) and/or the host side (bCheckHost) of the port should be checked. The function compares the user value (bUserAvail) and /or the hosts value (bHostAvail) if the corresponding Boolean flag is true. The ports are sorted by their sequence number. Depending on the port direction (bPinPortDir), the available ports are sorted in ascending (bPinPortDir = 0) or descending (bPinPortDir = 1) order.

API syntax

```
long AmGetNextAssetPort(handle hApiCnxBase, long lAssetId, long
lCabCnxTypeId, long lDutyId, long bCheckUser, long bCheckHost, long
bUserAvail, long bHostAvail, long bPinPortDir, long iPrevPortSeq);
```

Internal Basic syntax

```
Function AmGetNextAssetPort(lAssetId As Long, lCabCnxTypeId As Long,
lDutyId As Long, bCheckUser As Long, bCheckHost As Long, bUserAvail As
Long, bHostAvail As Long, bPinPortDir As Long, iPrevPortSeq As Long) As
Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lAssetId*: This parameter defines the device ID to search.
- *lCabCnxTypeId*: This parameter defines the cable connection type for the port.
- *lDutyId*: This parameter is the duty of the port.
- *bCheckUser*: This parameter is a flag to check the user side.
- *bCheckHost*: This parameter is a flag to check the host side.
- *bUserAvail*: This parameter defines the user side availability state to check.
- *bHostAvail*: This parameter defines the host side availability state to check.

- *bPinPortDir*: This parameter defines the pin direction to check.
  - 0=ascending
  - 1=descending
- *iPrevPortSeq*

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNextCableBundle()

The AmGetNextCableBundle API finds the next available bundle on a cable (lCableId) providing a given service (lDutyId) or no service at all. The status of the bundle must be "Available". Boolean flags are used to signify if the user side (bCheckUser) and/or the host side (bCheckHost) of the bundle should be checked. The function compares the user value (bUserAvail) and/ or the host value (bHostAvail) if the corresponding Boolean flag is true.

API syntax

```
long AmGetNextCableBundle(handle hApiCnxBase, long lCableId, long
lDutyId, long bCheckUser, long bCheckHost, long bUserAvail, long
bHostAvail);
```

Internal Basic syntax

```
Function AmGetNextCableBundle(lCableId As Long, lDutyId As Long,
bCheckUser As Long, bCheckHost As Long, bUserAvail As Long, bHostAvail As
Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the ID of the cable to check.
- *lDutyId*: This parameter defines the duty to locate.
- *bCheckUser*: This parameter states to check the user side connection of the bundle.
- *bCheckHost*: This parameter states to check the host side connection of the bundle.
- *bUserAvail*: This parameter defines the user side connection state to locate.
- *bHostAvail*: This parameter defines the host side connection state to locate.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNextCablePair()

The AmGetNextCablePair API finds the next available cable pair in a cable (lCableId) of a given type (lPairType). The pairs are sorted by cable's pair ID.

API syntax

```
long AmGetNextCablePair(handle hApiCnxBase, long lCableId, long
lPairTypeId, long iStartPairSeq);
```

Internal Basic syntax

```
Function AmGetNextCablePair(lCableId As Long, lPairTypeId As Long,
iStartPairSeq As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the ID of the cable to search.
- *lPairTypeId*: This parameter defines the cable pair type to locate.
- *iStartPairSeq*

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNTDomains()

This function enables you to get the domain of the user connected to the database.

API syntax

```
long AmGetNTDomains(char *pstrDomains, long lDomains);
```

Internal Basic syntax

```
Function AmGetNTDomains() As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNTMachinesInDomain()

This function enables you to get the list of computers in a domain as a column (computer names separated by commas). If the domain is empty, the function returns ERR_CANCEL(2), but the execution is not interrupted.

API syntax

```
long AmGetNTMachinesInDomain(char *strDomain, char *pstrMachines, long
lMachines, long bUseDC);
```

Internal Basic syntax

```
Function AmGetNTMachinesInDomain(strDomain As String, bUseDC As Long) As
String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDomain*: This parameter contains the name of the domain to explore.
- *bUseDC*: If this parameter is set to 1, the function queries the domain controller for a list of computers. If this parameter is set to 0 (the default value) the function uses the system function libraries to find the list of computers.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetNTUsersInDomain()

This function enables you to get the list of users of a domain. The list is returned as two columns (login,fullname). '|' is used as column separator, ',' as line separator.

API syntax

```
long AmGetNTUsersInDomain(char *strDomain, char *pstrUsers, long lUsers);
```

Internal Basic syntax

```
Function AmGetNTUsersInDomain(strDomain As String) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDomain*: This parameter contains the name of the domain to explore.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetPackageNames()

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

AmGetPOLinePrice()

This function enables you to calculate the price of an order line.

API syntax

```
double AmGetPOLinePrice(handle hApiCnxBase, long lPOrdLineId);
```

Internal Basic syntax

```
Function AmGetPOLinePrice(lPOrdLineId As Long) As Double
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdLineId*: This parameter contains the identifier of the order line.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetPOLinePriceCur()

This function enables you to find the currency code for the order line

API syntax

```
long AmGetPOLinePriceCur(handle hApiCnxBase, long lPOrdLineId, char
*pstrPrice, long lPrice);
```

Internal Basic syntax

```
Function AmGetPOLinePriceCur(lPOrdLineId As Long) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdLineId*: This parameter contains the identifier of the order line.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetPOLineReference()

This function enables you to get the catalog reference description corresponding to the purchase order line.

API syntax

```
long AmGetPOLineReference(handle hApiCnxBase, long lPOrdLineId, char
*pstrRef, long lRef);
```

Internal Basic syntax

```
Function AmGetPOLineReference(lPOrdLineId As Long) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdLineId*: This parameter contains the identifier of the order line.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetPrimaryTenant()

This function returns the primary tenant ID of the current user.

API syntax

```
long AmGetPrimaryTenant(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmGetPrimaryTenant() As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetRecordFromMainId()

This function returns the ID number of a record identified by a value of the primary key of the table containing this record.

API syntax

```
handle AmGetRecordFromMainId(handle hApiCnxBase, char *strTable, long
lId);
```

Internal Basic syntax

```
Function AmGetRecordFromMainId(strTable As String, lId As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strTable*: This parameter contains the SQL name of the table containing the record concerned by the operation.
- *lId*: This parameter contains the value of the primary key of the table for this records.

Notes

This function systematically returns a record handle, except when the table is not found. If no record is found in the specified table, an error is raised at each execution of a function using the handle returned by this function.

AmGetExistRecordFromMainId()

This function returns the ID number of a record identified by a value of the primary key of the table containing this record.

API syntax

```
handle AmGetExistRecordFromMainId(handle hApiCnxBase, char *strTable,
```

```
long lId);
```

Internal Basic syntax

```
Function AmGetExistRecordFromMainId(strTable As String, lId As Long) As
Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strTable*: This parameter contains the SQL name of the table containing the record concerned by the operation.
- *lId*: This parameter contains the value of the primary key of the table for this records.

Notes

This function systematically returns a record handle, except when the table is not found. If no record is found in the specified table, an error is raised at each execution of this function and NULL record object returned.

AmGetRecordHandle()

This function returns the handle of a record that is the current result of a query identified by its handle. This record can be used to write in the database. This function only works if the query contains the primary key of the record.

API syntax

```
handle AmGetRecordHandle(handle hApiQuery);
```

Internal Basic syntax

```
Function AmGetRecordHandle(hApiQuery As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle of a query object.

AmGetRecordId()

This function returns the ID number of a record identified by its handle. In the case of a record being inserted, this value is 0.

API syntax

```
long AmGetRecordId(handle hApiRecord);
```

Internal Basic syntax

```
Function AmGetRecordId(hApiRecord As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains a valid handle of the record whose ID number you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetRelDstField()

This function returns a handle on the target field of a link.

API syntax

```
handle AmGetRelDstField(handle hApiField);
```

Internal Basic syntax

```
Function AmGetRelDstField(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle on the link concerned by the operation.

AmGetRelSrcField()

This function returns a handle on the source field of a link.

API syntax

```
handle AmGetRelSrcField(handle hApiField);
```

Internal Basic syntax

```
Function AmGetRelSrcField(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle on the link concerned by the operation.

AmGetRelTable()

This function returns a handle on the relation table of an N-N link.

API syntax

```
handle AmGetRelTable(handle hApiField);
```

Internal Basic syntax

```
Function AmGetRelTable(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle on the link concerned by the operation.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmGetReverseLink()

This function returns the handle of the reverse link specified by the handle contained in the *hApiField* parameter.

API syntax

```
handle AmGetReverseLink(handle hApiField);
```

Internal Basic syntax

```
Function AmGetReverseLink(hApiField As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a handle of the link whose reverse link you want to know.

AmGetScreenSetsNames()

API syntax

```
long AmGetScreenSetsNames(handle hApiCnxBase, char *strSeparator, char *strDefaultVal, char *pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmGetScreenSetsNames(strSeparator As String, strDefaultVal As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetScriptValue()

API syntax

```
long AmGetScriptValue(handle hApiObject, char *strScriptName, char
*strObject, char *strPath);
```

Internal Basic syntax

```
Function AmGetScriptValue(hApiObject As Long, strScriptName As String,
strObject As String, strPath As String) As Variant
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetSelfFromMainId()

Returns the description string for a record in a given table.

API syntax

```
long AmGetSelfFromMainId(handle hApiCnxBase, char *strTableName, long
lId, char *pstrRecordDesc, long lRecordDesc);
```

Internal Basic syntax

```
Function AmGetSelfFromMainId(strTableName As String, lId As Long) As
String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: This parameter contains the SQL name of the table containing record concerned by the operation.
- *lId*: This parameter contains the ID number concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetSerialModifiedPages()

API syntax

```
long AmGetSerialModifiedPages(handle hApiCnxBase, char
*strScreenSqlNameAndSet, char *strSerialization, char *pstrResult, long
lResult);
```

Internal Basic syntax

```
Function AmGetSerialModifiedPages(strScreenSqlNameAndSet As String,
strSerialization As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetSerialNbFilters()

API syntax

```
long AmGetSerialNbFilters(handle hApiCnxBase, char *strSerialization);
```

Internal Basic syntax

```
Function AmGetSerialNbFilters(strSerialization As String) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetSourceTable()

Returns the handle of the source table of the link indicated in the *hApiField* parameter.

API syntax

```
handle AmGetSourceTable(handle hApiField);
```

Internal Basic syntax

```
Function AmGetSourceTable(hApiField As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle of the link whose source table you want to know.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmGetTable()

This function returns the handle of a table identified by its position (number) in the current connection.

API syntax

```
handle AmGetTable(handle hApiCnxBase, long lPos);
```

Internal Basic syntax

```
Function AmGetTable(lPos As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lPos*: This parameter contains the position of the table in the current connection. Its values are comprised between "0" and `AmGetTableCount`.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmGetTableCount()

This function returns the number of tables in the database concerned by the current connection.

API syntax

```
long AmGetTableCount(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmGetTableCount() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableDescription()

This function returns, as a character string ("String" format), the long description of a table identified by a handle.

API syntax

```
long AmGetTableDescription(handle hApiTable, char *pstrDesc, long lDesc);
```

Internal Basic syntax

```
Function AmGetTableDescription(hApiTable As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a valid handle of the table whose long description you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableFromName()

This function returns the handle of a table identified by its SQL name in the current connection.

API syntax

```
handle AmGetTableFromName(handle hApiCnxBase, char *strName);
```

Internal Basic syntax

```
Function AmGetTableFromName(strName As String) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strName*: This parameter contains the SQL name of the table whose handle you want to recover.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmGetTableLabel()

This function returns, as a character string ("String" format), the label of a table identified by a handle.

API syntax

```
long AmGetTableLabel(handle hApiTable, char *pstrLabel, long lLabel);
```

Internal Basic syntax

```
Function AmGetTableLabel(hApiTable As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiTable*: This parameter contains a valid handle of the table whose label you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableList()

API syntax

```
long AmGetTableList(handle hApiCnxBase, char *pstrXmlResult, long
lXmlResult);
```

Field of application

Version: 5.20

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableName()

Returns the SQL name of a table as a character string.

API syntax

```
long AmGetTableName(handle hApiTable, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetTableName(hApiTable As Long) As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiTable*: Valid handle of the table whose name you want to recover.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableRights()

This function returns, as a character string ("String" format), the users rights for a table given by a handle. The returned string consists of a maximum of two characters that indicate the status of creation and deletion rights:

- "c" indicates that the user has creation rights for the table.
- "d" indicates that the user has deletion rights on the table.

Thus, for example:

- " c" means that the user has creation rights for the table only.
- "cd" means that the user has both creation and deletion rights for the table.

API syntax

```
long AmGetTableRights(handle hApiTable, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetTableRights(hApiTable As Long) As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a valid handle of the table for which you want to know the user rights.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableSchema()

API syntax

```
long AmGetTableSchema(handle hApiCnxBase, char *strTableSqlName, char
*pstrXmlResult, long lXmlResult);
```

Field of application

Version: 5.20

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTableSqlName()

This function returns, as a character string ("String" format), the SQL name of a table identified by a handle.

API syntax

```
long AmGetTableSqlName(handle hApiTable, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmGetTableSqlName(hApiTable As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiTable`*: This parameter contains a valid handle of the table whose SQL name you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTargetTable()

Returns the SQL name of the target table of a link.

API syntax

```
handle AmGetTargetTable(handle hApiField);
```

Internal Basic syntax

```
Function AmGetTargetTable(hApiField As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: Handle of the link concerned by the operation.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmGetTrace()

The AmGetTrace API gets the trace between two nodes (lUserId, lHostId) in the cable link table. The trace direction (lTraceDir) identifies if the trace should be user-to-host (lTraceDir = 1) or host-to-user (lTraceDir = 0). The trace type (lTraceType) indicates if the trace is a connection (lTraceType = 1) or a disconnection (lTraceType = 2). The full trace indicator (bFullTrace) identifies if the trace include only modified nodes (bFullTrace = 0) or the entire trace (bFullTrace = 1).

API syntax

```
long AmGetTrace(handle hApiCnxBase, long lUserId, long lHostId, long
iTraceDir, long iTraceType, long bFullTrace, char *pstrTrace, long
lTrace);
```

Internal Basic syntax

```
Function AmGetTrace(lUserId As Long, lHostId As Long, iTraceDir As Long,
iTraceType As Long, bFullTrace As Long) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lUserId*: This parameter defines the starting connection link ID.
- *lHostId*: This parameter defines the ending connection link ID.

- *iTraceDir*: This parameter specifies the direction of the connection.
  - 0=host to user
  - 1=user to host
- *iTraceType*: This parameter defines the connection type.
  - 1=connection
  - 2=disconnection
- *bFullTrace*: This parameter specifies to ignore the partial trace and return the whole trace string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTraceFromHist()

The AmGetTraceFromHist API is for calculating a string from Trace History using Trace Operations to show new connectivity versus existing connectivity.

API syntax

```
long AmGetTraceFromHist(handle hApiCnxBase, long lProjTraceOutId, long
iTraceDir, char *strDelimiter, char *pstrTraceint, long lTraceint, long
bUpdateFlag);
```

Internal Basic syntax

```
Function AmGetTraceFromHist(lProjTraceOutId As Long, iTraceDir As Long,
strDelimiter As String, bUpdateFlag As Long) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lProjTraceOutId*: This parameter defines the project trace ID.
- *iTraceDir*: This parameter specifies the direction of the connection.
  - 0=host to user
  - 1=user to host
- *strDelimiter*: This parameter is the string delimiter to show existing connects and disconnects.
- *bUpdateFlag*: This parameter is an optional parameter to AmGetTraceHist API to update the amCabTraceOut.TraceString.
  - 0=false
  - 1=true

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetTypedLinkField()

Returns a handle of the field whose value is the SQL name of the target table of the typed link indicated in the *hApiField* parameter.

API syntax

```
handle AmGetTypedLinkField(handle hApiField);
```

Internal Basic syntax

```
Function AmGetTypedLinkField(hApiField As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: This parameter contains a valid handle of the typed link at the origin of the operation.

AmGetUserEnvSessionItem()

API syntax

```
long AmGetUserEnvSessionItem(handle hApiCnxBase, char *return, long
lreturn, char *strSection, char *strEntry);
```

Internal Basic syntax

```
Function AmGetUserEnvSessionItem(strSection As String, strEntry As
String) As String
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetVersion()

This function returns the build number of Asset Manager in the form of a character string.

API syntax

```
long AmGetVersion(char *pstrBuf, long lBuf);
```

Internal Basic syntax

```
Function AmGetVersion() As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetViewModifiedPages()

API syntax

```
long AmGetViewModifiedPages(handle hApiCnxBase, char *strViewSqlName,
char *pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmGetViewModifiedPages(strViewSqlName As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmGetViewNbFilters()

API syntax

```
long AmGetViewNbFilters(handle hApiCnxBase, char *strViewSqlName);
```

Internal Basic syntax

```
Function AmGetViewNbFilters(strViewSqlName As String) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHasAdminPrivilege()

This function returns "TRUE" (value other that 0) if the connected user has administration rights.

API syntax

```
long AmHasAdminPrivilege(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmHasAdminPrivilege() As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHasRelTable()

This function enables you to test whether a link has a relation table or not.

API syntax

```
long AmHasRelTable(handle hApiField);
```

Internal Basic syntax

```
Function AmHasRelTable(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a valid handle on the link concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHasRightsForCreation()

This function enables you to determine whether the connected user has creation rights for a given table.

Internal Basic syntax

```
Function AmHasRightsForCreation(strTable As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strTable*: This parameter contains the SQL name of the table concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal=amHasRightsForCreation("amEmplDept")
```

AmHasRightsForDeletion()

This function enables you to determine whether the connected user has deletion rights for a given table.

Internal Basic syntax

```
Function AmHasRightsForDeletion(strTable As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTable*: This parameter contains the SQL name of the table concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal=amHasRightsForDeletion("amEmplDept")
```

AmHasRightsForFieldUpdate()

This function enables you to determine whether the connected user has update rights for a given field.

Internal Basic syntax

```
Function AmHasRightsForFieldUpdate(strTable As String, strField As
String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTable*: This parameter contains the SQL name of the table concerned by the operation.

- *strField*: This parameter contains the SQL name of the field (the table is specified in the *strTable* parameter) concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal=amHasRightsForFieldUpdate("amEmplDept", "Location")
```

AmHelpdeskAddCheckCall()

This function adds a ticket history line documenting a call to verify the status of a ticket.

API syntax

```
long AmHelpdeskAddCheckCall(handle hApiCnxBase, long lTicketId, long
lCallDuration, long lContactId, long lAuthorId, char *strComment, char
*strResolutionCode, char *strSatisfactionLevel, long bAddToKnoledgeBase);
```

Internal Basic syntax

```
Function AmHelpdeskAddCheckCall(lTicketId As Long, lCallDuration As Long,
lContactId As Long, lAuthorId As Long, strComment As String,
strResolutionCode As String, strSatisfactionLevel As String,
bAddToKnoledgeBase As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.
- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text appended to the ticket description.
- *strResolutionCode*: This parameter contains the itemized list values denoting the resolution code for the ticket.
- *strSatisfactionLevel*: This parameter contains the itemized list values denoting the level of user satisfaction with the processing of the ticket.
- *bAddToKnoledgeBase*: This parameter contains a Boolean value indicating whether the ticket is to be integrated to the knowledge base.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskAddCloseCall()

This function adds a ticket history line for the activity that authorizes the closure of a helpdesk ticket.

API syntax

```
long AmHelpdeskAddCloseCall(handle hApiCnxBase, long lTicketId, long
lCallDuration, long lContactId, long lAuthorId, char *strComment, char
*strResolutionCode, char *strSatisfactionLevel, long bAddToKnoledgeBase);
```

Internal Basic syntax

```
Function AmHelpdeskAddCloseCall(lTicketId As Long, lCallDuration As Long,
lContactId As Long, lAuthorId As Long, strComment As String,
strResolutionCode As String, strSatisfactionLevel As String,
bAddToKnoledgeBase As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.

- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text that is appended to the ticket description.
- *strResolutionCode*: This parameter contains the itemized list values denoting the resolution code for the ticket.
- *strSatisfactionLevel*: This parameter contains the itemized list values denoting the level of user satisfaction with the processing of the ticket.
- *bAddToKnoledgeBase*: This parameter contains a Boolean value indicating whether the ticket is to be integrated to the knowledge base.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskAddEnteringCall()

This function adds a ticket history line for the activity that enters additional data.

API syntax

```
long AmHelpdeskAddEnteringCall(handle hApiCnxBase, long lTicketId, long
lCallDuration, long lContactId, long lAuthorId, char *strComment);
```

Internal Basic syntax

```
Function AmHelpdeskAddEnteringCall(lTicketId As Long, lCallDuration As
Long, lContactId As Long, lAuthorId As Long, strComment As String) As
Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.
- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text that is appended to the ticket description.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskAddOutgoingCall()

This function adds a ticket history line documenting an attempt to contact the customer.

API syntax

```
long AmHelpdeskAddOutgoingCall(handle hApiCnxBase, long lTicketId, long lCallDuration, long lContactId, long lAuthorId, char *strComment);
```

Internal Basic syntax

```
Function AmHelpdeskAddOutgoingCall(lTicketId As Long, lCallDuration As Long, lContactId As Long, lAuthorId As Long, strComment As String) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.
- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text that is appended to the ticket description.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskAddResumeCall()

This function resumes a suspended helpdesk ticket.

API syntax

```
long AmHelpdeskAddResumeCall(handle hApiCnxBase, long lTicketId, long lCallDuration, long lContactId, long lAuthorId, char *strComment);
```

Internal Basic syntax

```
Function AmHelpdeskAddResumeCall(lTicketId As Long, lCallDuration As Long, lContactId As Long, lAuthorId As Long, strComment As String) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.
- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text that is appended to the ticket description.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskAddSuspendCall()

This function adds a ticket history line for the activity that authorizes the suspension of a helpdesk ticket.

API syntax

```
long AmHelpdeskAddSuspendCall(handle hApiCnxBase, long lTicketId, long
lCallDuration, long lContactId, long lAuthorId, char *strComment, long
iSuspLimitType, long lSuspDurationHours, long tmSuspLimitDate, char
*strSuspType, long iFreezeEscalation, long lsuspActionId);
```

Internal Basic syntax

```
Function AmHelpdeskAddSuspendCall(lTicketId As Long, lCallDuration As
Long, lContactId As Long, lAuthorId As Long, strComment As String,
iSuspLimitType As Long, lSuspDurationHours As Long, tmSuspLimitDate As
```

```
Date, strSuspType As String, iFreezeEscalation As Long, lsuspActionId As
Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTicketId*: This parameter contains the identifier of the helpdesk ticket.
- *lCallDuration*: This parameter contains the duration of the call in seconds.
- *lContactId*: This parameter contains the identifier of the contact for the helpdesk ticket.
- *lAuthorId*: This parameter contains the identifier of the author of the current update.
- *strComment*: This parameter is free-form text that is appended to the ticket description.
- *iSuspLimitType*: This parameter contains the type of suspension used, 0 (Date), 1 (Duration), 2 (Manual), 3 (Next business day), 4 (Next business week), 5 (Next business month).
- *lSuspDurationHours*: This parameter contains the duration of the suspension in hours when 1 (Duration) is selected for **iSuspLimitType**.
- *tmSuspLimitDate*: This parameter contains the end date of the suspension for limit types (**iSuspLimitType**) 0 (Date), 2 (Manual), 3 (Next business day), 4 (Next business week) and 5 (Next business month).
- *strSuspType*: This parameter contains suspension information.
- *iFreezeEscalation*: This parameter is a Boolean determining whether escalation activities and alarms should be halted during the suspension.
- *lsuspActionId*: This parameter contains the identifier of the action which will be run when the ticket is resumed.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmHelpdeskCanCloseFile()

This function enables you to determine whether the connected user can close a helpdesk ticket or not.

Internal Basic syntax

```
Function AmHelpdeskCanCloseFile() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user can close a helpdesk ticket, the function returns the value "1".

AmHelpdeskCanProceed()

This function enables you to determine whether the connected user can proceed with a helpdesk ticket or not.

Internal Basic syntax

```
Function AmHelpdeskCanProceed() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user can proceed with a helpdesk ticket, the function returns thevalue "1".

AmHelpdeskCanSaveCall()

This function enables you to determine whether the connected user can save a helpdesk ticket or not.

Internal Basic syntax

```
Function AmHelpdeskCanSaveCall() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user can save a helpdesk ticket, the function returns the value "1".

AmImportDocument()

This function creates and imports a document from a file.

API syntax

```
long AmImportDocument(handle hApiCnxBase, long lDocObjId, char
*strTableName, char *strFileName, char *strCategory, char
*strDesignation);
```

Internal Basic syntax

```
Function AmImportDocument(lDocObjId As Long, strTableName As String,
strFileName As String, strCategory As String, strDesignation As String)
As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lDocObjId*: This parameter contains the value that will be stored in the lDocObjId field of the amDocument table.
- *strTableName*: This parameter contains the value that will be stored in the DocObjTable field of the amDocument table. In practice, it is the SQL name of the table containing the record to which the document is attached.
- *strFileName*: This parameter contains the name of the file to import.
- *strCategory*: This parameter contains the category of the document, as is appears in AssetCenter.
- *strDesignation*: This parameter contains the name of the document, as it appears in AssetCenter.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmImportReport()

This function enables you to import a SAP Crystal Report from a file. It is imported into an existing record in the **amReport** table.

Internal Basic syntax

```
Function AmImportReport(lReportId As Long, strFileName As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lReportId*: This parameter contains the identifier of the record in the table **amReport** in which the imported report will be stored.
- *strFileName*: This parameter contains the full path of the file containing the report to be imported.

Output parameters

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

AmIncrementLogLevel()

This function displays the *strMsg* message in a history window and creates a node in the final page of a wizard.

All the following messages appear in this node.

Internal Basic syntax

```
Function AmIncrementLogLevel(strMsg As String, iType As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strMsg*: This parameter contains the text of the message to be displayed.
- *iType*: This parameter defines the icon associated with the message. The possible values are "1" for an error, "2" for a warning, and "4" for information.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmInsertRecord()

This function inserts a record previously created in the database. Only those records created using the `AmCreateRecord` function can be inserted in the database. Records accessed using a query cannot be inserted.

API syntax

```
long AmInsertRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmInsertRecord(hApiRecord As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains a handle of the record you want to insert in the database.

Output parameters

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

AmInstantiateReqLine()

This function enables you to directly instantiate a given request line.

API syntax

```
long AmInstantiateReqLine(handle hApiCnxBase, long lRequestLineId, long
bFinal, long lPOrderLineId, double dQty);
```

Internal Basic syntax

```
Function AmInstantiateReqLine(lRequestLineId As Long, bFinal As Long,
lPOrderLineId As Long, dQty As Double) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestLineId*: This parameter contains the identifier of the request line.
- *bFinal*This parameter enables you to specify whether you want to finalize the assignment.
- *lPOrderLineId*: This parameter contains the identifier of the order line.
- *dQty*: This parameter contains quantity to instantiate.

Output parameters

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

Notes

The function enables you to create requested elements without going through the procurement cycle. If bFinal = FALSE, then the asset will be created with the status Awaiting receipt.

AmInstantiateRequest()

This function enables you to directly instantiate the full contents of a given request.

API syntax

```
long AmInstantiateRequest(handle hApiCnxBase, long lRequestId, long lMulFactor);
```

Internal Basic syntax

```
Function AmInstantiateRequest(lRequestId As Long, lMulFactor As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestId*: This parameter contains the identifier of the request.
- *lMulFactor*: This parameter enables you to specify the number of instantiations to perform.

Output parameters

- 0: Normal execution.

- Other than zero: Error code.

AmIsConnected()

This function tests whether the current connection is valid.

API syntax

```
long AmIsConnected(handle hApiCnxBase);
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsExistingPage()

API syntax

```
long AmIsExistingPage(handle hApiCnxBase, char *strTableName, char
*strModifiedPagesNames, char *pstrResult, long lResult);
```

Internal Basic syntax

```
Function AmIsExistingPage(strTableName As String, strModifiedPagesNames
As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsExistingScreen()

API syntax

```
long AmIsExistingScreen(handle hApiCnxBase, char *strScreenSqlName, char *strScreenSet);
```

Internal Basic syntax

```
Function AmIsExistingScreen(strScreenSqlName As String, strScreenSet As String) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsFieldForeignKey()

This function enables you to determine whether a field is an foreign key in the database.

API syntax

```
long AmIsFieldForeignKey(handle hApiField);
```

Internal Basic syntax

```
Function AmIsFieldForeignKey(hApiField As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a handle of the field to be identified.

Output parameters

- 1: The field is a foreign key.
- 0: The field is not a foreign key.

AmIsFieldIndexed()

This function enables you to determine whether a field is indexed or not.

API syntax

```
long AmIsFieldIndexed(handle hApiField);
```

Internal Basic syntax

```
Function AmIsFieldIndexed(hApiField As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a handle of the field to be identified.

Output parameters

- 1: The field is indexed.
- 0: The field is not indexed.

AmIsFieldPrimaryKey()

This function enables you to determine whether a field is an primary key in the database.

API syntax

```
long AmIsFieldPrimaryKey(handle hApiField);
```

Internal Basic syntax

```
Function AmIsFieldPrimaryKey(hApiField As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: This parameter contains a handle of the field to be identified.

Output parameters

- 1: The field is a primary key.
- 0: The field is not a primary key.

AmIsFilterModifInSerial()

API syntax

```
long AmIsFilterModifInSerial(handle hApiCnxBase, char
*strScreenSqlNameAndSet, char *strSerialization);
```

Internal Basic syntax

```
Function AmIsFilterModifInSerial(strScreenSqlNameAndSet As String,
strSerialization As String) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsFilterModifInView()

API syntax

```
long AmIsFilterModifInView(handle hApiCnxBase, char *strViewSqlName);
```

Internal Basic syntax

```
Function AmIsFilterModifInView(strViewSqlName As String) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsHelpdeskAdmin()

This function enables you to determine whether the connected user is the helpdesk administrator.

Internal Basic syntax

```
Function AmIsHelpdeskAdmin() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user is the helpdesk administrator, the function returns the value "1".

AmIsHelpdeskMember()

This function enables you to determine whether the connected user belongs to a helpdesk group or not.

Internal Basic syntax

```
Function AmIsHelpdeskMember() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user belongs to a helpdesk group, the function returns the value "1".

AmIsHelpdeskSuper()

This function enables you to determine whether the connected user is a helpdesk group supervisor or not.

Internal Basic syntax

```
Function AmIsHelpdeskSuper() As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the connected user is a helpdesk supervisor, the function returns the value "1".

AmIsLeveragedUser()

This function checks whether a designated user is a leveraged user or not.

API syntax

```
long AmIsLeveragedUser(handle hApiCnxBase, long lEmplDeptId);
```

Internal Basic syntax

```
Function AmIsLeveragedUser(lEmplDeptId As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lEmplDeptId*: This parameter is the User ID (lEmplDeptId) of the designated user.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsLink()

Determines whether the object identified by its handle is a link or a field.

API syntax

```
long AmIsLink(handle hApiField);
```

Internal Basic syntax

```
Function AmIsLink(hApiField As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiField`*: Handle of the object concerned by the operation.

Output parameters

- 1: The object is a link.

- 0: The object is a field.

AmIsModuleAuthorized()

This function enables you to determine whether the connected user has access to a given module of the application or not.

Internal Basic syntax

```
Function AmIsModuleAuthorized(strModuleName As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strModuleName*: This parameter contains the name of the module concerned by the operation. The following is the list of possible modules:
  - API: Dynamic functions library
  - Admin: Administration module
  - Barcode: Barcode inventory module
  - Cable: Cable management module
  - Chargeback: Chargeback module
  - Contract: Contract management module

- ESD: AssetCenter and software distribution tools integration module
- Finance: Financial management module
- Helpdesk: Helpdesk module
- InfraTools: Module not used
- ITAM: Portfolio management module
- Knowlix: Knowlix integration module
- Leasing: Leasing management module
- OVCM: AssetCenter and OpenView Configuration Manager Solution integration module
- Procurement: Procurement management module
- Reconc: Reconciliation module
- SAM: Software Asset Management module
- WebService: Web services
- Wizard: Wizard management module
- Workflow: Workflow scheme management module

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

Not all modules are available or can be enabled from the application. The availabilitydepends on the license you have acquired from Peregrine Systems, Inc.

AmIsMultiTenant()

This function checks whether a particular table is multi-tenant enabled or not.

API syntax

```
long AmIsMultiTenant(handle hApiCnxBase, char *strTableName);
```

Internal Basic syntax

```
Function AmIsMultiTenant(strTableName As String) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTableName*: This parameter contains the table SQL name.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsMultiTenantSystem()

This function checks whether the current database is multi-tenant enabled or not.

API syntax

```
long AmIsMultiTenantSystem(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmIsMultiTenantSystem() As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmIsTypedLink()

Determines if the object identified by its handle is a typed link or not.

API syntax

```
long AmIsTypedLink(handle hApiField);
```

Internal Basic syntax

```
Function AmIsTypedLink(hApiField As Long) As Long
```

Field of application

Version: 3.02

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiField*: Handle of the object concerned by the operation.

Output parameters

- 1: The object is a typed link.
- 0: The object is not a typed link.

AmLastError()

This function returns the last error code generated by the last function executed in the context of the corresponding connection.

API syntax

```
long AmLastError(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmLastError() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" above function (and optionally the "AmLastErrorMsg()" on the next page function) to find out if an error occurred (and obtain its associated message).

AmLastErrorMsg()

This function returns the last error message occurred in the current connection.

API syntax

```
long AmLastErrorMsg(handle hApiCnxBase, char *pstrBuffer, long lBuffer);
```

Internal Basic syntax

```
Function AmLastErrorMsg() As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on the previous page function (and optionally the "AmLastErrorMsg()" above function) to find out if an error occurred (and obtain its associated message).

AmListToString()

This function converts the result of a character string obtained via the `AmDbGetList` function to a character string that can be displayed in the same way as the `AmDbGetString` function.

API syntax

```
long AmListToString(char *return, long lreturn, char *strSource, char *strColSep, char *strLineSep, char *strIdSep);
```

Internal Basic syntax

```
Function AmListToString(strSource As String, strColSep As String, strLineSep As String, strIdSep As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`strSource`*: This parameter contains the character string to be converted.
- *`strColSep`*: This parameter contains the character used as column separator in the string to be converted.
- *`strLineSep`*: This parameter contains the character used as line separator in the string to be converted.

- *strIdSep*: This parameter contains the character used as identifier separator in the string to be converted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmLog()

This function displays the *strMessage* message in a history window.

Internal Basic syntax

```
Function AmLog(strMessage As String, iLogType As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strMessage*: This parameter contains the text of the message to be displayed.
- *iLogType*: This parameter defines the icon associated with the message. The possible values are "1" for an error, "2" for a warning, and "4" for information.

Output parameters

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

Example

```
AmLog("This is a message")
```

AmLoginId()

This function returns the identifier of the connected user.

API syntax

```
long AmLoginId(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmLoginId() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example defines the identifier of the connected user as the default value for a database field:

```
RetVal=AmLoginId()
```

AmLoginName()

This function returns the login name of the connected user.

API syntax

```
long AmLoginName(handle hApiCnxBase, char *return, long lreturn);
```

Internal Basic syntax

```
Function AmLoginName() As String
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example defines the login name of the connected user as the default value for a database field:

```
RetVal=AmLoginName()
```

AmMapSubReqLineAgent()

This function enables you to establish the possible links between the sub-lines of a request line and those of an order line.

API syntax

```
long AmMapSubReqLineAgent(handle hApiCnxBase, long lRequestLineId, long lPorderLineId);
```

Internal Basic syntax

```
Function AmMapSubReqLineAgent(lRequestLineId As Long, lPorderLineId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lRequestLineId*: This parameter contains the identifier of the request line.
- *lPorderLineId*: This parameter contains the identifier of the request line.

Output parameters

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

AmMoveCable()

The AmMoveCable API moves a cable (lCableId) from its current location to a given destination location (lToLoc). If the project (lProjectId) and work order (lWorkOrderId) have values, the cable is added to the project and work order with the comment contained in the given comment (strComment). This comment describes the action that will be performed on the cable (that is, "Move cable from here to there").

API syntax

```
long AmMoveCable(handle hApiCnxBase, long lCableId, long lToLocId, long lProjectId, long lWorkOrderId, char *strComment);
```

Internal Basic syntax

```
Function AmMoveCable(lCableId As Long, lToLocId As Long, lProjectId As Long, lWorkOrderId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the ID of the cable to move.
- *lToLocId*: This parameter defines the cable ID to move.

- *lProjectId*: This parameter is the project ID.
- *lWorkOrderId*: This parameter defines the work order ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

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

AmMoveDevice()

The AmMoveDevice API moves a device (lAssetId) from its current location to a given destination location (lToLoc). If the project (lProjectId) and work order (lWorkOrderId) have values, the device is added to the project and work order with the given comment (strComment). This comment describes the action that will be performed on the device (that is, "Move device from here to there").

API syntax

```
long AmMoveDevice(handle hApiCnxBase, long lDeviceId, long lToLocationId,
long lProjectId, long lWorkOrderId, char *strComment);
```

Internal Basic syntax

```
Function AmMoveDevice(lDeviceId As Long, lToLocationId As Long,
lProjectId As Long, lWorkOrderId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓

	Available
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lDeviceId*: This parameter defines the device ID that will be moved.
- *lToLocationId*: This parameter defines the device's new location.
- *lProjectId*: This parameter is the project ID.
- *lWorkOrderId*: This parameter defines the work order ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

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

AmMsgBox()

This function displays a dialog box containing a message.

Internal Basic syntax

```
Function AmMsgBox(strMessage As String, lMode As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strMessage*: This parameter contains the message displayed in the dialog box.
- *lMode*: This parameter contains the displayed dialog box type (O for a simple dialog box with an OK button, 1 for a dialog box with OK and Cancel, 2 for a dialog box with just Cancel).

Output parameters

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

Example

```
AmMsgBox("Move carried out")
```

AmNbLanguages()

API syntax

```
long AmNbLanguages(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmNbLanguages() As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmOpenConnection()

Creates a connection to an Asset Manager database. *strDataSource* must be a valid data source (data sources are listed in the Asset Manager connection box).

You can open several connections to the same database or to different databases.

API syntax

```
handle AmOpenConnection(char *strDataSource, char *strUser, char
*strPwd);
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strDataSource*: Name of the data source.
- *strUser*: User name for the connection.
- *strPwd*: Password of the specified user.

AmOpenScreen()

This function enables you to open a screen or a view in Asset Manager.

> **Note:** On the Asset Manager web client, if a screen is opened by amOpenScreen, there is no back button available. This is a limitation.

Internal Basic syntax

```
Function AmOpenScreen(strScreenId As String, strContext As String,
strFilter As String, iMode As Long, strBindField As String, bStayReadOnly
As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	

	Available
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strScreenId*: This parameter contains the SQL name of the view of the system or user screen you want to open (in this order of priority).
- *strContext*: This optional parameter contains the list of identifiers of the records selected in the list on opening the screen.
- *strFilter*: This parameter contains an AQL filter applied on the list on opening the screen.
- *iMode*: This parameter contains the mode in which the screen is opened: consultation, edit, and so on. The possible values are: 0 (Consultation only), 1 (Consultation), 2 (Modification in progress), 3 (Creation in progress), 4 (Duplication in progress), 5 (Addition in progress), 6 (Selection in progress).
- *strBindField*: This parameter enables you to open a screen with a filter and a mode for opening a linked window. It uses the SQL name of the source field or the value CurrentSrcChoice to use the current context.
- *bStayReadOnly*: This parameter enables you to open a screen in read-only mode. No modifications are allowed, regardless of the user rights.

> **Note:** If `iMode` is set to 3, a "create record" operation is triggered. For example, if `strScreenId` is set to `amEmployee` and `iMode` is set to 3, a page/frame is opened for the creation of a new employee. When `iMode` is set to 3, `strContext` and `strFilter` do not take effect.
>
> If `iMode` is set to 0 and `strContext` is set to an ID of a record, the detail frame shows the detailed information of this record.
>
> If `iMode` is set to 0 and `strFilter` is set to an AQL filter, the list frame shows the filtered records.
>
> `strScreenId` can set with a screen set (for example, full or simple). If it is not set, the default screen set will be used.

Output parameters

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

AmOpenScreenEx()

This function enables you to open a screen or a view in Asset Manager.

Internal Basic syntax

```
Function AmOpenScreenEx(strScreenId As String, strScreenSet As String,
strContext As String, strFilter As String, iMode As Long, strBindField As
String, bStayReadOnly As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strScreenId*: This parameter contains the SQL name of the view of the system or user screen you want to open (in this order of priority).
- strScreenSet: This parameter defines the screen set (for example, simple or full) of the screen. If this parameter is not set, the default screen set will be used.

- *strContext*: This optional parameter contains the list of identifiers of the records selected in the list on opening the screen.
- *strFilter*: This parameter contains an AQL filter applied on the list on opening the screen.
- *iMode*: This parameter contains the mode in which the screen is opened: consultation, edit, and so on. The possible values are: 0 (Consultation only), 1 (Consultation), 2 (Modification in progress), 3 (Creation in progress), 4 (Duplication in progress), 5 (Addition in progress), 6 (Selection in progress).
- *strBindField*: This parameter enables you to open a screen with a filter and a mode for opening a linked window. It uses the SQL name of the source field or the value CurrentSrcChoice to use the current context.
- *bStayReadOnly*: This parameter enables you to open a screen in read-only mode. No modifications are allowed, regardless of the user rights.

> **Note:** If `iMode` is set to 3, a "create record" operation is triggered. For example, if `strScreenId` is set to `amEmployee` and `iMode` is set to 3, a page/frame is opened for the creation of a new employee. When `iMode` is set to 3, `strContext` and `strFilter` do not take effect.
>
> If `iMode` is set to 0 and `strContext` is set to an ID of a record, the detail frame shows the detailed information of this record.
>
> If `iMode` is set to 0 and `strFilter` is set to an AQL filter, the list frame shows the filtered records.

Output parameters

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

AmOverflowTables()

This function returns the SQL names of the overflow tables of a given table.

API syntax

```
long AmOverflowTables(handle hApiCnxBase, char *strBasisTable, char
*strOverflowTables, long lOverflowTables);
```

Internal Basic syntax

```
Function AmOverflowTables(strBasisTable As String) As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strBasisTable*: This parameter contains the SQL name of the table concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The comma is used as the separator in the list returned by the function. If no overflow table exists for a given table, the function returns an empty string.

Example

The following example returns the overflow tables of the Portfolio Items table (amPortfolio):

```
RetVal = AmOverflowTables("amPortfolio")
```

The result of this example is:

```
amComputer,amSoftInstall,amPhone
```

AmPagePath()

This function returns a string containing the execution path of the wizard, that is, the list of pages browsed. Backward jumps are ignored.

Internal Basic syntax

```
Function AmPagePath() As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmProgress()

This function displays, in the final page of a wizard, a progress indicator representing a percentage.

Internal Basic syntax

```
Function AmProgress(iProgress As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *iProgress*: This parameter contains the percentage of completion (between 0 and 100) used to define size of the progress indicator.

Output parameters

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

Example

```
AmProgress(85)
```

This function displays a progress indicator representing 85% completion.

AmPurgeRecord()

This function destroys a record.

API syntax

```
long AmPurgeRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmPurgeRecord(hApiRecord As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record concerned by the operation.

Output parameters

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

Notes

- The processing of linked records depends on the type of the link. For OWN typelinks, linked records are processed identically. In the case of a DEFINE or NORMALlink, foreign keys of linked records are reset to 0 and the archival fields are populatedwith the identifier and description string of the archived record.
- This function is available for a record from an archival table or a standard table.

AmQueryAddGroupByField()

This function adds a GroupBy field.

API syntax

```
long AmQueryAddGroupByField(handle hApiQuery, const char *strSqlPath);
```

Internal Basic syntax

```
Function AmQueryAddGroupByField(hApiQuery As Long, strSqlPath As String)
As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *strSqlPath*: This parameter defines a field path.

Output parameters

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

AmQueryAddOrderByField()

This function adds an **Order by** field.

API syntax

```
long AmQueryAddOrderByField(handle hApiQuery, const char *strSqlPath, int bDesc);
```

Internal Basic syntax

```
Function AmQueryAddOrderByField(hApiQuery As Long, strSqlPath As String, bDesc As Integer) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *strSqlPath*: This parameter defines a field path.
- *bDesc*: This parameter defines the sort direction.

Output parameters

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

AmQueryAddSelectField()

This function adds a field to a query.

API syntax

```
long AmQueryResetFrom(handle hApiQuery, const char *strSqlPath);
```

Internal Basic syntax

```
Function AmQuerySetFrom(hApiQuery As Long, strSqlPath As String) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *strSqlPath*: This parameter defines a field path.

Output parameters

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

AmQueryAndWhereCond()

This function adds Where clause to a query.

API syntax

```
long AmQueryAndWhereCond(handle hApiQuery, const char *strCond);
```

Internal Basic syntax

```
Function AmQueryAndWhereCond(hApiQuery As Long, strCond As String) As
Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *strCond*: This parameter defines a where clause.

Output parameters

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

AmQueryCreate()

This function creates a query object in the current connection. This object can then be used to send AQL statements to the database server.

API syntax

```
handle AmQueryCreate(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmQueryCreate() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

AmQueryExec()

This function executes an AQL query. It returns the first result of the query. The next result can be obtained via the `AmQueryNext` function.

When the query sent by this function returns a "Memo" type field the result is limited to 255 characters.

API syntax

```
long AmQueryExec(handle hApiQuery, char *strQueryCommand);
```

Internal Basic syntax

```
Function AmQueryExec(hApiQuery As Long, strQueryCommand As String) As
Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiQuery`*: This parameter contains a valid handle of the query object to which the AQL statements are sent.
- *`strQueryCommand`*: This parameter contains the body of the AQL query as a string.

Output parameters

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

AmQueryGet()

This function executes an AQL query without a cursor (one single result). It only returns one single line of results.

API syntax

```
long AmQueryGet(handle hApiQuery, char *strQueryCommand);
```

Internal Basic syntax

```
Function AmQueryGet(hApiQuery As Long, strQueryCommand As String) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiQuery`*: This parameter contains a valid handle of the query object to which the AQL statements are sent.

- *strQueryCommand*: This parameter contains the body of the AQL query as a string.

Output parameters

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

AmQueryNext()

This function returns the result of a query executed beforehand using the `AmQueryExec` function.

API syntax

```
long AmQueryNext(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryNext(hApiQuery As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle of the query object to which the AQL

statements are sent.

Output parameters

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

AmQueryResetGroupBy()

This function removes all Group by fields.

API syntax

```
long AmQueryResetGroupBy(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryResetGroupBy(hApiQuery As Long) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.

Output parameters

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

AmQueryResetOrderBy()

This function removes all Order By fields.

API syntax

```
long AmQueryResetOrderBy(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryResetOrderBy(hApiQuery As Long) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiQuery`*: This parameter contains a valid handle on a query object.

Output parameters

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

AmQueryResetSelect()

This function removes all the select fields.

API syntax

```
long AmQueryResetSelect(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryResetSelect(hApiQuery As Long) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiQuery`*: This parameter contains a valid handle on a query object.

Output parameters

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

AmQueryResetWhereCond()

This function removes a Where clause.

API syntax

```
long AmQueryResetWhereCond(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryResetWhereCond(hApiQuery As Long) As Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiQuery`*: This parameter contains a valid handle on a query object.

Output parameters

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

AmQuerySetAddMainField()

This function enables you to send a query in a mode where the main field of the table is automatically added to the list of fields to be returned. This type of query never returns a null identifier record.

API syntax

```
long AmQuerySetAddMainField(handle hApiQuery, long bAddMainField);
```

Internal Basic syntax

```
Function AmQuerySetAddMainField(hApiQuery As Long, bAddMainField As Long)
As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.

- *bAddMainField*: This parameter can have one of two values:
  - True: The main field of the table is added,
  - False: The main field of the table is not added.

Output parameters

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

AmQuerySetFrom()

This function sets the queries from table.

API syntax

```
long AmQuerySetFrom(handle hApiQuery, const char *strSqlTableName);
```

Internal Basic syntax

```
Function AmQuerySetFrom(hApiQuery As Long, strSqlTableName As String) As
Long
```

Field of application

Version: 4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *strSqlTableName*: This parameter defines the table name to set

Output parameters

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

AmQuerySetFullMemo()

By default, when executing the `AmQueryExec` function, the query truncates Memo type fields to 254 characters. This function sends the query in a mode where Memo fields are recovered in full.

API syntax

```
long AmQuerySetFullMemo(handle hApiQuery, long bFullMemo);
```

Internal Basic syntax

```
Function AmQuerySetFullMemo(hApiQuery As Long, bFullMemo As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle on a query object.
- *bFullMemo*: This parameter can have one of two values:
  - True: The query returns the Memo field in full,
  - False: The query truncates Memo fields to 254 characters.

Output parameters

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

AmQueryStartTable()

This function returns a handle of the table concerned by a query identified by its handle.

API syntax

```
handle AmQueryStartTable(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryStartTable(hApiQuery As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle of a query object.

Output parameters

In case of error, this function returns a non-valid handle (zero).

AmQueryStop()

This function interrupts the execution of a query identified by its handle. This query must have been launched beforehand using the `AmQueryExec` function.

API syntax

```
long AmQueryStop(handle hApiQuery);
```

Internal Basic syntax

```
Function AmQueryStop(hApiQuery As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiQuery*: This parameter contains a valid handle of a query object.

Output parameters

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

AmReceiveAllPOLines()

This function receives all the items on an order line (takes delivery in full).

> Delivery lines are created by an agent when the transaction is committed. You cannot access them beforehand.

API syntax

```
long AmReceiveAllPOLines(handle hApiCnxBase, long lPOrdId, long
lDelivId);
```

Internal Basic syntax

```
Function AmReceiveAllPOLines(lPOrdId As Long, lDelivId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdId*: This parameter contains the identifier of the order line containing the items to be received.
- *lDelivId*: This parameter contains the identifier of the receiving slip used to receive all the items present on the order line.

Output parameters

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

AmReceivePOLine()

This function takes delivery of a certain quantity of items on an order line (takes delivery in part) and returns the identifier of the delivery line.

> The delivery lines are created by an agent as soon as the transaction iscommitted. You cannot access them until this is performed.

API syntax

```
long AmReceivePOLine(handle hApiCnxBase, long lPOrdLineId, long lDelivId,
double dQty);
```

Internal Basic syntax

```
Function AmReceivePOLine(lPOrdLineId As Long, lDelivId As Long, dQty As Double) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPOrdLineId*: This parameter contains the identifier of the purchase order line containing the items to be received.
- *lDelivId*: This parameter contains the identifier of the receiving slip used to receive a certain quantity of items present on the order line.
- *dQty*: This parameter contains the quantity of items on the order line to be received in the receiving slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRefreshAllCaches()

This function refreshes the caches used in Asset Manager.

API syntax

```
long AmRefreshAllCaches(handle hApiCnxBase);
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

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

AmRefreshLabel()

The AmRefreshLabel API refreshes the label string of a given record (lMainId) in a given table (strTableName).

API syntax

```
long AmRefreshLabel(handle hApiCnxBase, long lMainId, char *strTableName,
```

```
char *pstrLabel, long lLabel);
```

Internal Basic syntax

```
Function AmRefreshLabel(lMainId As Long, strTableName As String) As
String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lMainId*: This parameter defines the ID that will be refreshed.
- *strTableName*: This parameter defines the table name for the lMainId.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRefreshProperty()

Reevaluates the value of a property identified by the *strVarName* parameter. If this property uses a script, the script is executed again.

Otherwise the tree of dependencies is updated.

Internal Basic syntax

```
Function AmRefreshProperty(strVarName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strVarName*: Name of the property (of the wizard) that you want to reevaluate.

Output parameters

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

AmRefreshTraceHist()

The AmRefreshTraceHist API refreshes a complete project trace entry and also has an optional parameter to allow refreshing of "individual" trace history entries. If this parameter is not provided, the complete trace history will be refreshed.

API syntax

```
long AmRefreshTraceHist(handle hApiCnxBase, long lCabTraceOutId, long lTraceHistId);
```

Internal Basic syntax

```
Function AmRefreshTraceHist(lCabTraceOutId As Long, lTraceHistId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCabTraceOutId*: This parameter is the cable trace output ID.
- *lTraceHistId*: This parameter is an optional parameter to allow refreshing of "individual" trace history entries.

Output parameters

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

AmReleaseConnection()

The AmReleaseConnection API releases the AM session to the AM connection idle pool or close it if the idle pool is full. If the session is closed, all objects (queries, records, tables, fields, and so on) created within this connection are automatically destroyed, their handles become invalid, and the connection handle no longer exists.

API syntax

```
long AmReleaseConnection(handle hApiCnxBase);
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- **hApiCnxBase**: AM session handle get from AmGetConnection

Output parameters

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

AmReleaseHandle()

This function frees the handle and sub-handles of an object.

API syntax

```
long AmReleaseHandle(handle hApiObject);
```

Internal Basic syntax

```
Function AmReleaseHandle(hApiObject As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *hApiObject*: This parameter contains a handle of the object concerned.

Output parameters

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

AmRemoveCable()

The AmRemoveCable API removes a cable (lCableId) from its current location. The status of the cable is updated to "Unavailable". If the project (lProjectId) and work order (lWorkOrderId) have values, the cable is added to the project and work order with comment contained in the given comment (strComment). This comment describes the action that will be performed on the cable (that is, "Remove cable from its current location").

API syntax

```
long AmRemoveCable(handle hApiCnxBase, long lCableId, long lProjectId,
long lWorkOrderId, char *strComment);
```

Internal Basic syntax

```
Function AmRemoveCable(lCableId As Long, lProjectId As Long, lWorkOrderId
As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter is the ID of the cable to remove.
- *lProjectId*: This parameter is the project ID.
- *lWorkOrderId*: This parameter defines the work order ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

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

AmRemoveDevice()

The AmRemoveDevice API removes a device (lAssetId) from its current location. The status of the device is updated to "Unavailable". If the project (lProjectId) and work order (lWorkOrderId) have values, the device is added to the project and work order with the given comment (strComment). This comment describes the action that will be performed on the device (that is, "Remove device from its current location").

API syntax

```
long AmRemoveDevice(handle hApiCnxBase, long lDeviceId, long lProjectId,
long lWorkOrderId, char *strComment);
```

Internal Basic syntax

```
Function AmRemoveDevice(lDeviceId As Long, lProjectId As Long,
lWorkOrderId As Long, strComment As String) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lDeviceId*: This parameter defines the device ID to remove.
- *lProjectId*: This parameter is the project ID.
- *lWorkOrderId*: This parameter defines the work order ID.
- *strComment*: This parameter is the comment that will be used on the work order.

Output parameters

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

AmResetPassword()

API syntax

```
long AmResetPassword(handle hApiCnxBase, char *strOldPassword, char
*strNewPassword);
```

Internal Basic syntax

```
Function AmResetPassword(strOldPassword As String, strNewPassword As
String) As Long
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

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

AmResetUserEnvSession()

API syntax

```
long AmResetUserEnvSession(handle hApiCnxBase, char *strSection);
```

Internal Basic syntax

```
Function AmResetUserEnvSession(strSection As String) As Long
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

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

AmResetUserPassword()

API syntax

```
long AmResetUserPassword(handle hApiCnxBase, char *strUser, char
*strPasswd, char *strNewPasswd);
```

Internal Basic syntax

```
Function AmResetUserPassword(strUser As String, strPasswd As String,
strNewPasswd As String) As Long
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRestoreRecord()

This function restores an archived record.

API syntax

```
long AmRestoreRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmRestoreRecord(hApiRecord As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	

	Available
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record concerned by the operation.

Output parameters

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

Notes

The processing of linked records depends on the type of the link. For OWN typelinks, linked records are processed identically. In the case of a DEFINE or NORMALlink, foreign keys of linked records are reset to 0 and the archival fields are populatedwith the identifier and description string of the archived record.

This function is available for a record from an archival table.

AmReturnAsset()

This function enables you to return an asset.

API syntax

```
long AmReturnAsset(handle hApiCnxBase, long lAstId, long lReturnId, long bCanMerge);
```

Internal Basic syntax

```
Function AmReturnAsset(lAstId As Long, lReturnId As Long, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lAstId*: This parameter contains the identifier of the asset to return.
- *lReturnId*: This parameter contains the identifier of the return slip.
- *bCanMerge*: This parameter enables you to specify whether the return can be merged with an existing line in the return slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmReturnContract()

This function enables you to return a contract.

API syntax

```
long AmReturnContract(handle hApiCnxBase, long lCntrId, long lReturnId,
long bCanMerge);
```

Internal Basic syntax

```
Function AmReturnContract(lCntrId As Long, lReturnId As Long, bCanMerge
As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lCntrId*: This parameter contains the identifier of the contract to return.
- *lReturnId*: This parameter contains the identifier of the return slip.
- *bCanMerge*: This parameter enables you to specify whether the return can be merged with an existing line in the return slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmReturnPortfolioItem()

This function enables you to return a portfolio item.

API syntax

```
long AmReturnPortfolioItem(handle hApiCnxBase, long lPfId, double dQty,
long lFromRecptLineId, long lReturnId, long bCanMerge);
```

Internal Basic syntax

```
Function AmReturnPortfolioItem(lPfId As Long, dQty As Double,
lFromRecptLineId As Long, lReturnId As Long, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lPfId*: This parameter contains the identifier of the portfolio item to return.
- *dQty*: This parameter contains the quantity (in the unit of the model) to return.

- *lFromRecptLineId*: This parameter contains the identifier of the source receipt line.
- *lReturnId*: This parameter contains the identifier of the return slip.
- *bCanMerge*: This parameter enables you to specify whether the return can be merged with an existing line in the return slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmReturnTraining()

This function enables you to return a training.

API syntax

```
long AmReturnTraining(handle hApiCnxBase, long lTrainingId, long
lReturnId, long bCanMerge);
```

Internal Basic syntax

```
Function AmReturnTraining(lTrainingId As Long, lReturnId As Long,
bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTrainingId*: This parameter contains the identifier of the training to return.
- *lReturnId*: This parameter contains the identifier of the return slip.
- *bCanMerge*: This parameter enables you to specify whether the return can be merged with an existing line in the return slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmReturnWorkOrder()

This function enables you to return a work order.

API syntax

```
long AmReturnWorkOrder(handle hApiCnxBase, long lWOId, long lReturnId,
long bCanMerge);
```

Internal Basic syntax

```
Function AmReturnWorkOrder(lWOId As Long, lReturnId As Long, bCanMerge As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lWOId*: This parameter contains the identifier of the work order to return.
- *lReturnId*: This parameter contains the identifier of the return slip.
- *bCanMerge*: This parameter enables you to specify whether the return can be merged with an existing line in the return slip.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRevCryptPassword()

This function encrypts a reversible password. The function that allows decryption of a password encrypted with this function is not available.

API syntax

```
long AmRevCryptPassword(handle hApiCnxBase, char *return, long lreturn,
char *strPassword);
```

Internal Basic syntax

```
Function AmRevCryptPassword(strPassword As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strPassword*: This parameter contains the password to encrypt.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRgbColor()

This function gives the RGB value of the color corresponding to the *strText* parameter.

API syntax

```
long AmRgbColor(char *strText);
```

Internal Basic syntax

```
Function AmRgbColor(strText As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strText*: This parameter contains the name of the color:
  - White
  - ltGray
  - Gray

- Dkgray
- Black
- Red
- Green
- Blue
- Yellow
- Cyan
- Magenta
- Dkyellow
- Dkgreen
- Dkcyan
- Dkblue
- Dkmagenta
- Dkred

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmRollback()

This function cancels all modifications made before the declaration of the start of the transaction (performed via the `AmStartTransaction` function).

API syntax

```
long AmRollback(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmRollback() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmSetFieldDateOnlyValue()

This function modifies a field in a record. This function does not update the database. The modification will be made when the record is updated or inserted, or when the transaction is committed.

API syntax

```
long AmSetFieldDateOnlyValue(handle hApiRecord, char *strFieldName, long
dtptmValue);
```

Internal Basic syntax

```
Function AmSetFieldDateOnlyValue(hApiRecord As Long, strFieldName As
String, dtptmValue As Date) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiRecord`*: This parameter contains the handle of the record containing the field to be modified.
- *`strFieldName`*: This parameter contains the SQL name of the field to be modified.
- *`dtptmValue`*: This parameter contains the new value of the field in "Date" format only. Unlike the `AmSetFieldDateValue` function, only the Date part is processed, the Time part is omitted.

Output parameters

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

AmSetFieldDateValue()

This function modifies a field in a record. This function does not update the database. The modification will be made when the record is updated or inserted, or when the transaction is committed.

API syntax

```
long AmSetFieldDateValue(handle hApiRecord, char *strFieldName, long
tmValue);
```

Internal Basic syntax

```
Function AmSetFieldDateValue(hApiRecord As Long, strFieldName As String,
tmValue As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record containing the field to be modified.
- *strFieldName*: This parameter contains the SQL name of the field to be modified.
- *tmValue*: This parameter contains the new value of the field in "Date" format.

Output parameters

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

AmSetFieldDoubleValue()

This function modifies a field in a record. This function does not update the database.

API syntax

```
long AmSetFieldDoubleValue(handle hApiRecord, char *strFieldName, double dValue);
```

Internal Basic syntax

```
Function AmSetFieldDoubleValue(hApiRecord As Long, strFieldName As String, dValue As Double) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the handle of the record containing the field to be modified.
- *strFieldName*: This parameter contains the SQL name of the field to be modified.
- *dValue*: This parameter contains the new value of the field in "Double" format.

Output parameters

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

AmSetFieldLongValue()

This function modifies a field in a record. This function does not update the database. To modify the value of a date, time or date+time date you must express the new value in terms of seconds elapsed since 01/01/1970 at 00:00:00.

API syntax

```
long AmSetFieldLongValue(handle hApiRecord, char *strFieldName, long
lValue);
```

Internal Basic syntax

```
Function AmSetFieldLongValue(hApiRecord As Long, strFieldName As String,
lValue As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiRecord`*: This parameter contains the handle of the record containing the field to be modified.
- *`strFieldName`*: This parameter contains the SQL name of the field to be modified.
- *`lValue`*: This parameter contains the new value of the field.

Output parameters

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

AmSetFieldStrValue()

This function modifies a field in a record. This function does not update the database.

API syntax

```
long AmSetFieldStrValue(handle hApiRecord, char *strFieldName, char
*strValue);
```

Internal Basic syntax

```
Function AmSetFieldStrValue(hApiRecord As Long, strFieldName As String,
strValue As String) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *`hApiRecord`*: This parameter contains the handle of the record containing the field to be modified.

- *strFieldName*: This parameter contains the SQL name of the field to be modified.
- *strValue*: This parameter contains the new value of the field in "String" format.

Output parameters

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

AmSetLinkFeatureValue()

This function sets the value of a link type feature for a given record.

API syntax

```
long AmSetLinkFeatureValue(handle hApiRecord, char *strFeatSqlName, char
*strDstSelfValue, long lDstId);
```

Internal Basic syntax

```
Function AmSetLinkFeatureValue(hApiRecord As Long, strFeatSqlName As
String, strDstSelfValue As String, lDstId As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains the identifier of the record to which the link type feature is associated.
- *strFeatSqlName*: This parameter contains the SQL name of the link type feature whose value you want to set. This SQL name is always preceded by "fv_".
- *strDstSelfValue*: This parameter contains the value of the feature as it will be displayed for the record. It is the "Self" value of the record with identifier *lDstId*. If you pass an invalid or non-existent value, you take the risk of corrupting the integrity of the database.
- *lDstId*: This parameter contains the identifier of the record to which the link type feature points.

Output parameters

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

AmSetPrimaryTenant()

This function sets the primary tenant ID of the current user.

API syntax

```
long AmSetPrimaryTenant(handle hApiCnxBase, long lTenantId);
```

Internal Basic syntax

```
Function AmSetPrimaryTenant(lTenantId As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *lTenantId*: This parameter can either be a new primary tenant ID obtained from the user's viewable tenant list, or the shared tenant ID if the user is authorized to edit the shared data.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmSetProperty()

This function sets the value of a property identified by its name. It also updates the tree of dependencies of this property.

Internal Basic syntax

```
Function AmSetProperty(strVarName As String, vValue As Variant) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strVarName*: This parameter contains the name of the property whose value you want to set.
- *vValue*: This parameter contains the new value for the property.

Output parameters

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

AmSetUserEnvSessionItem()

API syntax

```
long AmSetUserEnvSessionItem(handle hApiCnxBase, char *strSection, char
*strEntry, char *strValue);
```

Internal Basic syntax

```
Function AmSetUserEnvSessionItem(strSection As String, strEntry As
String, strValue As String) As Long
```

Field of application

Version: 4.4.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

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

AmSetVisibleTenants()

This function selects the visible objects to a user based on the user's tenant mode.

Internal Basic syntax

```
Function AmSetVisibleTenants(iFlag As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iFlag*: This parameter denotes the tenant mode ID, 0 (Base mode, that is Context or Primary with Shared), 1 (Shared only), 2 (Context or Primary), 3 (All viewables).

Output parameters

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

Notes

Context or Primary indicates that if an action has a context, the user is set to the tenant of the context record; otherwise the primary tenant. **All Viewables** indicates that all viewable tenants of the current user.

AmShowCableCrossConnect()

This function displays the cross-connections screen.

Internal Basic syntax

```
Function AmShowCableCrossConnect(lCableId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lCableId*: This parameter contains the identifier of the cable concerned by the operation.

Output parameters

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

AmShowDeviceCrossConnect()

This function displays the cross-connections screen for a cable device.

Internal Basic syntax

```
Function AmShowDeviceCrossConnect(lDeviceId As Long) As Long
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lDeviceId*: This parameter contains the identifier of the cable device concerned by the operation.

Output parameters

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

AmSqlTextConst()

This function transforms a string to be used in a query. The following operations are performed on the string:

- All single quotes (') are doubled,
- Single quotes are added at the start and end of the string.

API syntax

```
long AmSqlTextConst(char *return, long lreturn, char *str);
```

Internal Basic syntax

```
Function AmSqlTextConst(str As String) As String
```

Field of application

Version: 4.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *str*: This parameter contains the character string to process.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim strReq as String
strReq="SELECT lEmplDeptId FROM amEmplDept WHERE Name=" & amSqlTextConst(strName)
```

This query is valid, even if the strName variable contains single quotes.

AmStandIn()

This function returns the identifier of the employee standing in for the employee with the identifier *lEmployeeId* on the date *tmDate*.

API syntax

```
long AmStandIn(handle hApiCnxBase, long lEmployeeId, long tmDate);
```

Internal Basic syntax

```
Function AmStandIn(lEmployeeId As Long, tmDate As Date) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`lEmployeeId`*: This parameter contains the identifier of the employee whose stand-in you want to know.
- *`tmDate`*: This parameter contains the date on which the function performs the search.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If on the specified date, `tmDate`, the employee with identifier `lEmployeeId` is present, the function returns their identifier.If the employee is absent and no stand-in is designated, the function returns 0.

If the employee is absent and no stand-in is designated, the function returns 0.

Example

```
If [User.Parent.Supervisor] = 0 Then
  RetVal = amStandIn([User], amDate())
  if RetVal = 0 Then RetVal = [User]
Else
  RetVal = amStandIn([User.Parent.Supervisor], amDate())
  if RetVal = 0 Then RetVal = [User.Parent.Supervisor]
```

```
End If
```

AmStandInGroup()

This function returns the identifier of the employee group standing in for the employee with the identifier *lEmployeeId* on the date *tmDate*.

API syntax

```
long AmStandInGroup(handle hApiCnxBase, long lEmployeeId, long tmDate);
```

Internal Basic syntax

```
Function AmStandInGroup(lEmployeeId As Long, tmDate As Date) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lEmployeeId*: This parameter contains the identifier of the employee whose stand-in group you want to know.
- *tmDate*: This parameter contains the date on which the function performs the search.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If on the specified date, `tmDate`, the employee with identifier `lEmployeeId` is present, the function returns 0.

If the employee is absent and no stand-in group is designated, the function also returns 0.

AmStartTransaction()

This function starts a new transaction with the database associated with the connection. The next "Commit" or "Rollback" statement will validate or cancel all the modifications made to the database.

API syntax

```
long AmStartTransaction(handle hApiCnxBase);
```

Internal Basic syntax

```
Function AmStartTransaction() As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmStartup()

This function must be applied before all other functions. It initializes calls to the Asset Manager library.

API syntax

```
void AmStartup();
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

AmTableDesc()

This function generates a character string with the format "<Description of the table> (<SQL name of the table>)" from the SQL name of the table.

API syntax

```
long AmTableDesc(handle hApiCnxBase, char *return, long lreturn, char
```

```
*strSqlName);
```

Internal Basic syntax

```
Function AmTableDesc(strSqlName As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strSqlName*: SQL name of the table for which a description string is required. If this parameter contains an invalid SQL name, the function returns a question mark ("?").

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example generates a description string for the table of assets (SQL name: amAsset):

```
AmTableDesc("amAsset")
```

The result is as follows:

```
Assets (amAsset)
```

AmTaxRate()

This function calculates a tax rate according to a tax type, tax jurisdiction and a date.

API syntax

```
double AmTaxRate(handle hApiCnxBase, char *strTaxRateName, long
lTaxLocId, long tmDate, double dValue);
```

Internal Basic syntax

```
Function AmTaxRate(strTaxRateName As String, lTaxLocId As Long, tmDate As
Date, dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strTaxRateName*: This parameter contains the SQL name of the tax type used to calculate the tax rate.
- *lTaxLocId*: This parameter contains the ID number of the tax jurisdiction concerned by the tax type.

- *tmDate*: This parameter contains the date for which you want to know the tax rate.
- *dValue*: Obsolete parameter, kept for compatibility reasons. Set this parameter to the value of your choice.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmTransferSerialFilterToQueryTable()

API syntax

```
long AmTransferSerialFilterToQueryTable(handle hApiCnxBase, char
*strScreenSqlNameAndSet, char *strSerialization, char *strFuncDomain,
char *strNewQuerySqlName, char *strNewQueryName, long
bWithParamInstantiation, long bOverWrite);
```

Internal Basic syntax

```
Function AmTransferSerialFilterToQueryTable(strScreenSqlNameAndSet As
String, strSerialization As String, strFuncDomain As String,
strNewQuerySqlName As String, strNewQueryName As String,
bWithParamInstantiation As Long, bOverWrite As Long) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmTransferSerialPropsToScreen()

API syntax

```
long AmTransferSerialPropsToScreen(handle hApiCnxBase, char
*strScreenSqlNameAndSet, char *strSerialization, char
*strToScreenSqlName, char *strToScreenDesc, char *strToScreenSet, char
*strToFuncDomain, char *strFilterPage, char *strModifiedPagesNames, char
*strPackageBaseName, char *strLang, long bVisibleInNavigationTree, long
bOverWrite);
```

Internal Basic syntax

```
Function AmTransferSerialPropsToScreen(strScreenSqlNameAndSet As String,
strSerialization As String, strToScreenSqlName As String, strToScreenDesc
As String, strToScreenSet As String, strToFuncDomain As String,
strFilterPage As String, strModifiedPagesNames As String,
strPackageBaseName As String, strLang As String, bVisibleInNavigationTree
As Long, bOverWrite As Long) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmTransferViewFilterToQueryTable()

API syntax

```
long AmTransferViewFilterToQueryTable(handle hApiCnxBase, char
*strViewSqlName, char *strFuncDomain, char *strNewQuerySqlName, char
*strNewQueryName, long bWithParamInstantiation, long bOverWrite);
```

Internal Basic syntax

```
Function AmTransferViewFilterToQueryTable(strViewSqlName As String,
strFuncDomain As String, strNewQuerySqlName As String, strNewQueryName As
String, bWithParamInstantiation As Long, bOverWrite As Long) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmTransferViewPropsToScreen()

API syntax

```
long AmTransferViewPropsToScreen(handle hApiCnxBase, char
*strSourceViewSqlName, char *strToScreenSqlName, char *strToScreenDesc,
char *strToScreenSet, char *strToFuncDomain, char *strFilterPage, char
*strModifiedPagesNames, char *strPackageBaseName, char *strLang, long
bVisibleInNavigationTree, long bOverWrite);
```

Internal Basic syntax

```
Function AmTransferViewPropsToScreen(strSourceViewSqlName As String,
strToScreenSqlName As String, strToScreenDesc As String, strToScreenSet
As String, strToFuncDomain As String, strFilterPage As String,
strModifiedPagesNames As String, strPackageBaseName As String, strLang As
String, bVisibleInNavigationTree As Long, bOverWrite As Long) As Long
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Output parameters

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

AmUpdateDetail()

This function is used in the data-entry wizards. The context (table for which a record is updated or populated or updated using the wizard) is therefore clearly defined. The function updates or populates fields or links of the context according to a value. This function not allowed in non-modal wizards.

Internal Basic syntax

```
Function AmUpdateDetail(strFieldName As String, varValue As Variant) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	

	Available
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *strFieldName*: This parameter contains the SQL name of the feature to be updated.
- *varValue*: This parameter contains the new value of the field.

Output parameters

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

AmUpdateLossLines()

This function enables you to update all loss values for contracts using the loss-value rule referenced by the *lLossValId* identifier.

Internal Basic syntax

```
Function AmUpdateLossLines(lLossValId As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lLossValId*: This parameter contains the identifier of the loss-value rule.

Output parameters

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

AmUpdateRecord()

This function enables you to update a record.

API syntax

```
long AmUpdateRecord(handle hApiRecord);
```

Internal Basic syntax

```
Function AmUpdateRecord(hApiRecord As Long) As Long
```

Field of application

Version: 2.52

	Available
AssetManager API	✓
Configuration script of a field or link	
"Script" type action	✓
Wizard script	
FINISH.DO script of a wizard	✓

Input parameters

- *hApiRecord*: This parameter contains a handle of the record containing the field to be updated.

Output parameters

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

AmUpdateUser()

This information updates the information (domain, NT user name, description) concerning an employee in the database.

API syntax

```
long AmUpdateUser(handle hApiCnxBase, long lId, char *strNTDomain, char *strNTUserName, char *strNTUserDesc);
```

Internal Basic syntax

```
Function AmUpdateUser(lId As Long, strNTDomain As String, strNTUserName As String, strNTUserDesc As String) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lId*: This parameter contains the identifier of the concerned user in the database.
- *strNTUserName*: This parameter contains the NT user name of the employee.
- *strNTDomain*: This parameter contains the NT domain name of the employee.
- *strNTUserDesc*: This parameter contains the description associated with the NT user.

Output parameters

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

AmValidateTIR()

This function performs a TIR (Tenant Integrity Rule) verification on a designated record of the current context table, mainly used in the validity script. It returns '1' if all fields/links of a record passes the TIR verification; '0' if the TIR verification fails in any of its fields/links.

Internal Basic syntax

```
Function AmValidateTIR(lMainId As Long) As Long
```

Field of application

Version: 9.30

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *lMainId*: This parameter is the main ID of the record. The value can be '0' if you are checking the current selected record from the context table.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

AmValueOf()

Used in a wizard, this function returns the value of the property identified by the *strVarName* parameter.

Internal Basic syntax

```
Function AmValueOf(strVarName As String) As Variant
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strVarName*: This parameter contains the name of the property whose value we want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example returns the value of the "Page1.Label" property:

```
AmValueOf("Page1.Label")
```

Use this function with care because it breaks the dependency string of the property being processed.

AmWizChain()

This function executes a wizard B, inside a wizard A. When wizard B has finished executing, wizard A takes over again.

Internal Basic syntax

```
Function AmWizChain(strWizSqlName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	

	Available
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strWizSqlName*: SQL name of the wizard to be executed.

Output parameters

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

AmWorkItemCompleteEvt

This function forces transition after a manual operation (Wizard) is performed.

Internal Basic syntax

```
Function AmWorkItemCompleteEvt(lWfItemId As Long,strContextTable As
String, lActivityId As Long,lWfInstanceId As Long ) As Long
```

Field of application

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lWfItemId*: This parameter contains the identifier of a workflow item.
- *strContextTable*: This parameter contains the name of the context table.
- *lActivityId*: This parameter contains the identifier of the activity.
- *lWfInstanceId*: This parameter contains the identifier of the workflow instance.

Output parameters

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

AmWorkTimeSpanBetween()

This function returns the duration of working periods between two dates. This duration is expressed in seconds; it respects the information in a calendar of working periods.

API syntax

```
long AmWorkTimeSpanBetween(handle hApiCnxBase, char *strCalendarSqlName,
long tmEnd, long tmStart);
```

Internal Basic syntax

```
Function AmWorkTimeSpanBetween(strCalendarSqlName As String, tmEnd As
Date, tmStart As Date) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	✓

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strCalendarSqlName*: This parameter contains the SQL name of the calendar of working periods used to calculate the duration of the working period between the two dates. If this parameter is omitted, the calculated duration does not take working periods into account.
- *tmEnd*: This parameter contains the end date for the period used in calculating the working period.
- *tmStart*: This parameter contains the start date for the period used in calculating the working period.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example calculates the working period between 01/09/1998 at 08:00 and 24/09/1998 at 19:00. The calendar used, whose SQL name is "Calendar_Paris", defines the following working periods:

- From Monday to Thursday from 08:00 to 12:00, and then from 14:00 to 18:00.
- Fridays from 08:00 to 12:00, and then from 14:00 to 17:00.

```
AmWorkTimeSpanBetween("Calendar_Paris", "1998/09/24 19:00:00", "1998/09/01
08:00:00")
```

This example returns the value 507,600 which represents the number of working seconds between the two dates.

AppendOperand()

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

strExpr strOperator strOperand

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

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

ApplyNewVals()

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

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Asc()

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

Internal Basic syntax

```
Function Asc(strAsc As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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

Internal Basic syntax

```
Function Atn(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

BasicToLocalDate()

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

Internal Basic syntax

```
Function BasicToLocalDate(strDateBasic As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

BasicToLocalTime()

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

Internal Basic syntax

```
Function BasicToLocalTime(strTimeBasic As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

BasicToLocalTimeStamp()

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

Internal Basic syntax

```
Function BasicToLocalTimeStamp(strTSBasic As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Beep()

Plays a beep on the machine.

Internal Basic syntax

```
Function Beep()
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

CDbl()

Converts an expression to a "Double".

Internal Basic syntax

```
Function CDbl(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *dValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Chr()

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

Internal Basic syntax

```
Function Chr(lChr As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lChr*: ASCII code of the character.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

Converts any valid expression to an Integer.

Internal Basic syntax

```
Function CInt(iValue As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

CLng()

Converts any valid expression to a Long.

Internal Basic syntax

```
Function CLng(lValue As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Cos()

Returns the cosine of a number, expressed in radians.

Internal Basic syntax

```
Function Cos(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The conversion formula for degrees to radians is the following:

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

Example

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

CountOccurences()

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

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

CountValues()

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

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim MyStr
```

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

CSng()

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

Internal Basic syntax

```
Function CSng(fValue As Single) As Single
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *fValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

CStr()

Converts any valid expression to a String.

Internal Basic syntax

```
Function CStr(strValue As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *strValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

CurDir()

Returns the current path.

Internal Basic syntax

```
Function CurDir() As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

CVar()

Converts any valid expression to a Variant.

Internal Basic syntax

```
Function CVar(vValue As Variant) As Variant
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *vValue*: Expression to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Date()

Returns the current system date.

Internal Basic syntax

```
Function Date() As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

DateAdd()

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

Internal Basic syntax

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

Field of application

Version: 2.51

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

DateAddLogical()

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

Internal Basic syntax

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

Field of application

Version: 2.51

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

DateDiff()

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

Internal Basic syntax

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

Field of application

Version: 2.51

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓

	Available
FINISH.DO script of a wizard	✓

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 (such as 1800).
- *iMonth*: Month.
- *iDay*: Day.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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 (that is, 1-31 for days, 1-12 for months, and so on), the function returns an empty date.

DateValue()

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

Internal Basic syntax

```
Function DateValue(tmDate As Date) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example:

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

Returns the value:

```
1999/09/24
```

Day()

Returns the day contained in the *tmDate* parameter.

Internal Basic syntax

```
Function Day(tmDate As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

EnumToComboBox()

This function reorganizes the items of a free itemized list so that they are in a format compatible with the wizard list-control. This enables you to display the values of free itemized lists in drop-down lists in wizards.

Internal Basic syntax

```
Function EnumToComboBox(strFormat As String) As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`strFormat`*: This parameter contains the list of entries of the system itemized list. It is better if this parameter contains the result of execution of the `AmDbGetList()` function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example takes the values of the amWOPriority free itemized list and reorganizes them in a format that is compatible with the wizard list-control:

```
Dim strValues As String
  strValues = AmDbGetList("SELECT Value FROM amItemListVal WHERE
ItemizedList.Identifier = 'amWOPriority'", "", ",", "")
  RetVal = EnumToComboBox(strValues)
```

EscapeSeparators()

Prefixes one or more separator characters with an escape character.

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ExeDir()

This function returns the full path of the executable.

Internal Basic syntax

```
Function ExeDir() As String
```

Field of application

Version: 3.60

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Exp()

Returns the exponent of a number.

Internal Basic syntax

```
Function Exp(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ExtractValue()

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.

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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" in the
source string
  MyStr=ExtractValue("you", ",", "\")  :'Returns "you" and leaves "" in the source
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()

Copies a file or a folder.

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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()

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

Internal Basic syntax

```
Function FileDateTime(strFileName As String) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

FileExists()

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

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

Internal Basic syntax

```
Function FileExists(strFileName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

FileLen()

Returns the size of a file.

Internal Basic syntax

```
Function FileLen(strFileName As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Fix()

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

Internal Basic syntax

```
Function Fix(dValue As Double) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

FormatDate()

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

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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.

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
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example:

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

returns "Iyouheweyouthey".

FV()

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

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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()

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

Internal Basic syntax

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

Field of application

Version: 3.2.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

GetListItem()

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

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example:

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

returns "is".

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

returns "a".

GetUserRightsList()

This function allows you to query user rights in the current database based on the selected table or field.

API syntax

```
int GetUserRightsList(const CString& strTableName, const CString&
strSqlName, byte bType, byte bRight, int bUserRole, CString* pstrList);
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- strTableName: The SQL name of the table which you want to query.
- strSqlName: The SQL name of the field/link/feature/calculate field which you want to query. When you query the user right for a table, this parameter is ignored.
- bUserRole: Indicates if you want to query user role (true) or user profile (false).
- bType: It indicates that if strSqlName is a field, link, feature, or calculate field.
  - 0x1: FIELD_RIGHT_TYPE
  - 0x2: LINK_RIGHT_TYPE
  - 0x4: FEATURE_RIGHT_TYPE
  - 0x8: CALCFIELD_RIGHT_TYPE
  - 0x10: TABLE_RIGHT_TYPE
- bRight: The rights on the object you want to query. The object can be a field, link, feature, calculated field, or table.
  - Field rights:
    - 0x01: RIGHT_READ
    - 0x02: RIGHT_CWRITE
    - 0x04: RIGHT_UWRITE
  - Table rights:
    - 0x20: RIGHT_CREATE
    - 0x40: RIGHT_DELETE

Output parameters

- pstrList: The queried user role or user profile separated by ",".

Hex()

Returns the hexadecimal value of a decimal parameter.

Internal Basic syntax

```
Function Hex(dValue As Double) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Hour()

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

Internal Basic syntax

```
Function Hour(tmTime As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

InStr()

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

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

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

Example

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

Int()

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

Internal Basic syntax

```
Function Int(dValue As Double) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

IPMT()

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

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
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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()

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

Internal Basic syntax

```
Function IsNumeric(strString As String) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

If the string contains a numeric value, the function returns the value -1. Otherwise,it returns the value 0.

A string is evaluated as being a numeric value if it contains a combination of the following characters:

- Numbers 0 to 9
- Characters + - . e E
- Space
- Tab
- Page break
- Carriage return

Kill()

Deletes a file.

Internal Basic syntax

```
Function Kill(strKilledFile As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓

	Available
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

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

LCase()

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

Internal Basic syntax

```
Function LCase(strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
' This example uses the LTrim and RTrim functions to strip leading ' and trailing
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()

Returns the left most iNumber characters of a string parameter.

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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.

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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.

Internal Basic syntax

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

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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

Internal Basic syntax

```
Function Len(vValue As Variant) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *vValue*: Variant concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

LocalToBasicDate()

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

Internal Basic syntax

```
Function LocalToBasicDate(strDateLocal As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

LocalToBasicTime()

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

Internal Basic syntax

```
Function LocalToBasicTime(strTimeLocal As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

LocalToBasicTimeStamp()

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

Internal Basic syntax

```
Function LocalToBasicTimeStamp(strTSLocal As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

LocalToUTCDate()

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

Internal Basic syntax

```
Function LocalToUTCDate(tmLocal As Date) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Log()

Returns the natural log of a number.

Internal Basic syntax

```
Function Log(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

LTrim()

Removes all leading spaces in a string.

Internal Basic syntax

```
Function LTrim(strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strString*: Character string to process.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
' This example uses the LTrim and RTrim functions to strip leading ' and trailing
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()

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

Internal Basic syntax

```
Function MakeInvertBool(lValue As Long) As Long
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *lValue*: Number concerned by the operation.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Mid()

Returns a substring within a string.

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim strTest as String
  strTest="One Two Three"      :' Defines the test string
```

```
strTest=Mid(strTest,5,3)     :' strTest="Two"
RetVal=strTest
```

Minute()

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

Internal Basic syntax

```
Function Minute(tmTime As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

MkDir()

Creates a new directory.

Internal Basic syntax

```
Function MkDir(strMkDirectory As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

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()

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

Internal Basic syntax

```
Function Month(tmDate As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Name()

Changes the name of file.

Internal Basic syntax

```
Function Name(strSource As String, strDest As String)
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

- *strDest*: New file name.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

Returns the current date and time.

Internal Basic syntax

```
Function Now() As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

NPER()

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

Internal Basic syntax

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

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *`1`* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

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

Oct()

Returns the octal value of the decimal parameter.

Internal Basic syntax

```
Function Oct(dValue As Double) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ParseDate()

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

Internal Basic syntax

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

Field of application

Version: 3.6.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ParseDMYDate()

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

```
dd/mm/yyyy
```

Internal Basic syntax

```
Function ParseDMYDate(strDate As String) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	

	Available
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDate*: Date stored as a string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ParseMDYDate()

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

`mm/dd/yyyy`

Internal Basic syntax

```
Function ParseMDYDate(strDate As String) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDate*: Date stored as a string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

ParseYMDDate()

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

Internal Basic syntax

```
Function ParseYMDDate(strDate As String) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strDate*: Date stored as a string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

PMT()

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

Internal Basic syntax

```
Function PMT(dblRate As Double, iNper As Long, dblPV As Double, dblFV As Double, iType As Long) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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()

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.

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
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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()

This function returns the actual amount of an annuity based on constant and periodic future deadlines, and on a fixed interest rate.

Internal Basic syntax

```
Function PV(dblRate As Double, iNper As Long, dblPmt As Double, dblFV As
Double, iType As Long) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, at the start of the period)

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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.

QueryListAllUserRights()

This function allows you to query all user rights in the current database.

API syntax

```
int QueryListAllUserRights(CString *pstrBuffer);
```

Field of application

Version: 4.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Output parameters

- pstrBuffer: All user rights in the current database serialized in XML format.

Randomize()

Initializes the random number generator.

Internal Basic syntax

```
Function Randomize(lValue As Long)
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

See also:

- "Rnd()" on page 589

Example

```
Dim MyNumber
  Randomize
  MyNumber= Int((10*Rnd)+1)    :'Returns a random value between 1 and 10.
  RetVal=MyNumber
```

RATE()

This function returns the interest rate per date of payment for an annuity.

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
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 (that is, at the end of the period)
  - *1* if the payments are due in advance (that is, 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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

- 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()

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.

Internal Basic syntax

```
Function RemoveRows(strList As String, strRowNames As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

See also:

- "SubList()" on page 606
- "SetSubList()" on page 595
- "ApplyNewVals()" on page 486

Example

```
Dim MyStr
  MyStr=RemoveRows("a1|a2=a0,b1|b2=b0", "a0,c0")       :'Returns "b1|b2=b0"
  RetVal=MyStr
```

Replace()

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.

Internal Basic syntax

```
Function Replace(strData As String, strOldPattern As String,
strNewPattern As String, bCaseSensitive As Long) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

Returns the rights most iNumber characters of the string parameter.

Internal Basic syntax

```
Function Right(strString As String, lNumber As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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.

Internal Basic syntax

```
Function RightPart(strFrom As String, strSep As String, bCaseSensitive As Long) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

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.

Internal Basic syntax

```
Function RightPartFromLeft(strFrom As String, strSep As String,
bCaseSensitive As Long) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

This function deletes all items (files and folders) from a folder. The folder itself is not deleted.

Internal Basic syntax

```
Function RmAllInDir(strRmDirectory As String, bStopIfError As Long) As
Long
```

Field of application

Version: 3.4.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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()

Removes an existing directory.

Internal Basic syntax

```
Function RmDir(strRmDirectory As String) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

Input parameters

- *`strRmDirectory`*: Full path of the directory to be removed.

Output parameters

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

Notes

The directory to be deleted must be empty. Otherwise, the function will not work.

Example

```
RetVal = RmDir("c:      mp")
```

Rnd()

Returns a value containing a random number.

Internal Basic syntax

```
Function Rnd(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

Before calling this function, you must use the **Randomize** function, without parameters, to initialize the random number generator.
See also:

- "Randomize()" on page 576

Example

```
Dim MyNumber
  Randomize
  MyNumber= Int((10*Rnd)+1)    :'Returns a random value between 1 and 10.
  RetVal=MyNumber
```

RoundValue()

This function calculates the rounding value of a number to the number of digits after the decimal point as specified by the *iDigits* parameter.

Internal Basic syntax

```
Function RoundValue(dValue As Double, iDigits As Long) As Double
```

Field of application

Version: 3.4.0

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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()

Removes all trailing spaces in a string.

Internal Basic syntax

```
Function RTrim(strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strString*: String to process.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
' This example uses the LTrim and RTrim functions to strip leading ' and trailing
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()

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

Internal Basic syntax

```
Function Second(tmTime As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim strSecond
  strSecond=Second(Date())
  RetVal=strSecond     :'Returns the number of seconds elapsed in the current hour, for
example "30" if the time is 15:45:30
```

SetMaxInst()

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.

API syntax

```
long SetMaxInst(long lMaxInst);
```

Internal Basic syntax

```
Function SetMaxInst(lMaxInst As Long) As Long
```

Field of application

Version: 4.3.0

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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

If you set the `lMaxInst` parameter to "0", the number of instructions that a scriptcan execute is unlimited.

SetSubList()

Defines the values of a sublist for a "ListBox" control.

Internal Basic syntax

```
Function SetSubList(strValues As String, strRows As String, strRowFormat
As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim MyStr
  MyStr=SetSubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "A2|A1=a0, B2|B1=b0",
"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()

Returns a value indicating the sign of a number.

Internal Basic syntax

```
Function Sgn(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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()

Launches an executable program.

Internal Basic syntax

```
Function Shell(strExec As String, bShowWindow As Long, bBackground As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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 program is executed asynchronously. If this parameter is set to 0, the function waits for the end of execution of the program before giving you back control (synchronous execution).

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim MyId
  MyId=Shell("C:\WinNT\explorer.exe")
  RetVal=""
```

Sin()

Returns the sine of an number that is expressed in radians.

Internal Basic syntax

```
Function Sin(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The conversion formula for degrees to radians is the following:

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

Example

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

Space()

Creates a string including the number of spaces indicated by the *iSpace* parameter.

Internal Basic syntax

```
Function Space(iCount As Long) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iCount*: Number of spaces to be inserted into the string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

This function can be used to format strings or to delete date in fixed length strings.

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
```

Sqr()

Returns the square root of a number.

Internal Basic syntax

```
Function Sqr(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *dValue*: Number whose square root you want to know.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim dCalc as Double
  dCalc=Sqr(81)
  RetVal=dCalc
```

Str()

Converts a number to a string.

Internal Basic syntax

```
Function Str(strValue As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strValue*: Number to convert to a string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

StrComp()

Compares two strings.

Internal Basic syntax

```
Function StrComp(strString1 As String, strString2 As String,
iOptionCompare As Long) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	
FINISH.DO script of a wizard	

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()

String returns a string consisting of the *strString* character repeated over and over *iCount* times.

Internal Basic syntax

```
Function String(iCount As Long, strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iCount*: Number of occurrences of the character *strString*.
- *strString*: Character used to compose the string.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim iCount as Integer
Dim strTest as String
  strTest="T"
  iCount=5
  RetVal=String(iCount,strTest)
```

SubList()

Returns a sublist of a list of values contained in a string representing the values of a "ListBox" control.

Internal Basic syntax

```
Function SubList(strValues As String, strRows As String, strRowFormat As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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")   :'Returns
"=a0,=b0,=c0"
  MyStr=SubList("a1|a2|a3=a0,b1|b2|b3=b0,c1|c2|c3=c0", "z0", "*=0")    :'Returns ""
```

```
  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=""
```

SysEnumToComboBox()

This function reorganizes the items of a system itemized list so that they are in a format compatible with the wizard list-control. This enables you to display the values of system itemized lists in drop-down lists in wizards.

Internal Basic syntax

```
Function SysEnumToComboBox(strFormat As String) As String
```

Field of application

Version: 4.3.0

	Available
AssetManager API	
Configuration script of a field or link	
"Script" type action	
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`strFormat`*: This parameter contains the list of entries of the system itemized list. It is better if this parameter contains the result of execution of the `AmGetFieldFormat()` function.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example takes the values of the seStatus system itemized list in the amWorkOrder table and reorganizes them in a format that is compatible with the wizard list-control:

```
Dim strFormat As String
  strFormat = AmGetFieldFormat(AmGetFieldFromName(AmGetTableFromName
("amWorkOrder"), "seStatus"))
  RetVal = SysEnumToComboBox(strFormat)
```

Tan()

Returns the tangent of a number expressed in radians.

Internal Basic syntax

```
Function Tan(dValue As Double) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Notes

The conversion formula for degrees to radians is the following:

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

Example

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

Time()

Returns the current time.

Internal Basic syntax

```
Function Time() As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal = Time()
```

Timer()

Returns the number of seconds elapsed since 12:00 AM.

Internal Basic syntax

```
Function Timer() As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal = Timer()
```

TimeSerial()

This function returns a time formatted according to the *iHour*, *iMinute* and *iSecond parameters*.

Internal Basic syntax

```
Function TimeSerial(iHour As Long, iMinute As Long, iSecond As Long) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *iHour*: Hour.
- *iMinute*: Minutes.
- *iSecond*: Seconds.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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 (that is, 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()

This function returns the time portion of a "Date+Time" value.

Internal Basic syntax

```
Function TimeValue(tmTime As Date) As Date
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.

- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example:

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

Returns the value:

```
15:00:00
```

ToSmart()

This function reformats a source string by capitalizing the first letter of each word.

Internal Basic syntax

```
Function ToSmart(strString As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *`strString`*: Source string to reformat.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The following example:

```
RetVal = ToSmart ("hello world")
```

returns the value:

```
Hello World
```

Trim()

Returns a copy of a string with the leading and trailing spaces removed.

Internal Basic syntax

```
Function Trim(strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓

	Available
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strString*: String to process.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
' This example uses the LTrim and RTrim functions to strip leading ' and trailing
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()

Returns a copy of a sting in which all lowercase characters are converted to uppercase.

Internal Basic syntax

```
Function UCase(strString As String) As String
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
' This example uses the LTrim and RTrim functions to strip leading ' and trailing
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()

Deletes all the escape characters from a string.

Internal Basic syntax

```
Function UnEscapeSeparators(strSource As String, strEscChar As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strSource*: Character string to process.
- *strEscChar*: Escape character to be deleted.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

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

Union()

Merges two strings delimited by separators. Duplicates are deleted.

Internal Basic syntax

```
Function Union(strListOne As String, strListTwo As String, strSeparator
As String, strEscChar As String) As String
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

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|b2,a1|a3\,b1|b2"
  RetVal=""
```

UTCToLocalDate()

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

Internal Basic syntax

```
Function UTCToLocalDate(tmUTC As Date) As Date
```

Field of application

Version: 3.5

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *tmUTC*: Date in UTC format.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
RetVal = UTCToLocaldate([DateTime])
```

Val()

Converts a string representing a number to a double.

Internal Basic syntax

```
Function Val(strString As String) As Double
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

- *strString*: Character string to convert.

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim strYear
Dim dYear as Double
  strYear=Year(Date())
 dYear=Val(strYear)
  RetVal=dYear :'Returns the current year
```

WeekDay()

Returns the day of the week contained in the date expressed by the *tmDate* parameter.

Internal Basic syntax

```
Function WeekDay(tmDate As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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()

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

API syntax

```
long XmlAttribute(char *return, long lreturn, char *strName, char
*strValue);
```

Internal Basic syntax

```
Function XmlAttribute(strName As String, strValue As String) As String
```

Field of application

Version: 5.10

	Available
AssetManager API	✓
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

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, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

The example below:

```
RetVal = XmlAttribute("Equation & condition","ten < eleven")
```

Returns the value:

```
Equation & condition = "ten &lt; eleven"
```

Year()

Returns the year contained in the value expressed by the *tmDate* parameter.

Internal Basic syntax

```
Function Year(tmDate As Date) As Long
```

Field of application

Version: 3.00

	Available
AssetManager API	
Configuration script of a field or link	✓
"Script" type action	✓
Wizard script	✓
FINISH.DO script of a wizard	✓

Input parameters

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

Output parameters

In case of error, there are two possibilities:

- In Asset Manager, the script containing the function is suspended and an error message issued to the user.
- If calling from an external program, you must call the "AmLastError()" on page 387 function (and optionally the "AmLastErrorMsg()" on page 388 function) to find out if an error occurred (and obtain its associated message).

Example

```
Dim strYear
  strYear=Year(Date())
  RetVal=strYear       :'Returns the current year
```

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Programmer reference (Asset Manager 9.62)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to ovdoc-ITSM@hpe.com.

We appreciate your feedback!