Script Outline

Script is the language for coding the procedures for installing tools in the PowerFRAMEVIEW.

To install a tool in the PowerFRAMEVIEW, use script language to code the processing corresponding to the messages. Specify the coded script name as a tool name in the PowerFRAMEVIEW Environment Setting. Now, the specified tool can be operated by sending and receiving messages in the same way as the tools corresponding to the PowerFRAMEVIEW.

When a tool is installed in the powerFRAMEVIEW, the Macro Manager is started by sending the START Message to the Tool Class in which the tool corresponding to the PowerFRAMEVIEW is installed. The macro manager reads the script and executes the Exec Function.

Start the tool by coding a script with one of the following methods:

- Start the tool in the exec function.

  Start the tool by calling the ToolExec or ToolExecEX in the exec function.

- Start the tool when the START message is received.

  Call the MakeEvent in the exec function so that the START message can be received.

  Start the tool by calling the ToolExec or ToolExecEX in the Callback Function specified in the MakeEvent.

  Use either of the above methods to install all tools in the PowerFRAMEVIEW by coding a script corresponding to the tool to be installed. Use the following methods to install a tool with script:

- Only start or stop the tool. (not corresponded to the message.)
- Make the tool correspond to the message.

Script Functions

The script functions are as follows:

- Tool start function

  Any tool can be started by coding script.
- Tool stop function

  The started tool can be stopped by coding script.
- Installation function to PowerFRAMEVIEW

  Tools can be installed in PowerFRAMEVIEW by coding the processing to be performed when a message is received.
- Built-in functions

  Various built-in functions are provided in script.   These functions   can be called from the script.

Script Features

The script features are as follows:

- Simple
  Script can be easily created, corrected, executed, and debugged.
- Various functions
  Various built-in functions such as character string operations are provided.
- Using the Japanese language
  Scripts can be written in Japanese.
- Control
  Control scripts similar to the C language can be written.

Script Terminology

The following special terms are used for explaining the script:

- Object

  Each entity executing a script or being executed with a script is called an object.   For example, tools started from script or dialog boxes created with scripts are all objects.

- Object ID

  A unique identifier for each object is assigned on a script.

  Using object IDs enables execution on the specific object from a script or data transfer so that objects can be operated and controlled.

- Script

  Script is the script language itself or anything that is coded with the script language.

- Function

  Function is the processing unit written with a script.

  Some functions are prepared with a script.

- Block

  A block is a unit for coding a script.   A group of statements and variable definitions is enclosed with '{' and '}'.

- Variable

  A variable is the area for storing data.   Each variable has a unique name.   The user-defined variable in script and the System Variable prepared with script are used.

Script Example for Installing a Tool in the PowerFRAMEVIEW

Script examples for installing tools in PowerFRAMEVIEW are as follows:
See the System Variable for "Script.Errordisplay" in the script example.

```
public :
   var tool ;
   var ToolClass = "EDIT";
   var Str_Filespec ;
   var Com_Filespec ;
   var Com_Term = 0 ;

exec() {
   var Filespec ;
   var Ret ;

   MessageOpen(ToolClass) ;
   Filespec = MakeFilespec("*", 0) ;
   Str_Filespec = Filespec ;

   MakeEvent("R", ToolClass, "START", Filespec, "start") ;

}

start() {
   var Get_MessageID ;
   var Make_MessageID;
   var Filespec ;
   var Ret ;
   var PathName ;
   var FileName ;
   var CommandLine ;

   var SenderToolClass ;
   var MessageType ;
   var ToolClassName ;
   var CommandName ;
   var Data ;
   var Originator ;

   Get_MessageID   = GetMessageID();
   Make_MessageID = MakeMessageID();
```

```
  Filespec = GetFilespec();
  Com_Filespec = Filespec ;

  SenderToolClass = GetSenderToolClass() ;
  MessageType       = GetMessageType() ;
  ToolClassName     = GetToolClass() ;
  CommandName       = GetCommand() ;
  Data                  = GetData(1) ;
  Originator         = GetOriginator();

  DeleteEvent("R", ToolClass, "START", Str_Filespec) ;
  MakeEvent("R", ToolClass, "START", Filespec, "start") ;

  MakeEvent("R", ToolClass, "STOP"        , Filespec, "stop" ) ;
  MakeEvent("R", ToolClass, "MAXIMIZE" , Filespec, "maximize" ) ;
  MakeEvent("R", ToolClass, "MINIMIZE" , Filespec, "minimize" ) ;
  MakeEvent("R", ToolClass, "NORMALIZE", Filespec, "normalize" ) ;

  PathName = FilespecToPathName(Filespec) ;
  CommandLine = cat("option parameter", PathName) ;

  tool = ToolExec("SAMPLE.EXE", CommandLine);
  tool = ToolExecEX("SAMPLE.EXE", CommandLine);

  if( IsNULL(tool) ) {
    MessageSend(Get_MessageID,   "F", ToolClass, "START" , Filespec);
    MessageSend(Make_MessageID, "N", ToolClass, "STATUS",
                      Filespec, "NOTRUNNING") ;
    del_event();
    Com_Term = 1 ;
    MessageClose();
    Close();
    return(0);
  }

  MessageSend(Make_MessageID, "N", ToolClass, "STATUS", Filespec, "STARTING") ;
  MessageSend(Make_MessageID, "N", ToolClass, "STATUS", Filespec, "READY") ;
  MessageSend(Get_MessageID, "N", ToolClass, "START", Filespec) ;
}

stop() {
  var Filespec ;
  var Ret;
  var Get_MessageID;
  var Make_MessageID;
  var ToolID ;

  Script.ErrorDisplay = False ;
  ToolID = closewindow(tool);
  Script.ErrorDisplay = True ;

  Get_MessageID = GetMessageID();
  Make_MessageID = MakeMessageID();

  Filespec = GetFilespec();
  MessageSend(Make_MessageID, "N", ToolClass, "STATUS",
                  Filespec, "NOTRUNNING") ;
  MessageSend(Get_MessageID,   "N", ToolClass, "STOP", Filespec) ;

  del_event() ;
  MessageClose();
  Com_Term = 1 ;
  Close();
}

maximize() {
  var MessageID ;
  var Filespec ;

  MaximizeWindow(tool) ;

  MessageID = GetMessageID();
  Filespec = GetFilespec();

  MessageSend(MessageID, "N", ToolClass, "MAXIMIZE", Filespec);
}

minimize() {
  var MessageID ;
  var Filespec ;

  MinimizeWindow(tool) ;
  MessageID = GetMessageID();
  Filespec = GetFilespec();

  MessageSend(MessageID, "N", ToolClass, "MINIMIZE", Filespec);
}

normalize() {
  var MessageID ;
  var Filespec ;

  RestoreWindow(tool) ;

  MessageID = GetMessageID();
  Filespec = GetFilespec();

  MessageSend(MessageID, "N", ToolClass, "NORMALIZE", Filespec);
}

Terminate() {
  var Ret ;
  var ToolID ;
  var Make_MessageID ;

  if(Com_Term != 0) {
     Return(0);
  }

  Script.ErrorDisplay = False ;
  ToolID = closewindow(tool);
  Script.ErrorDisplay = True ;

  Make_MessageID = MakeMessageID();
  MessageSend(Make_MessageID, "N", ToolClass, "STATUS",
                  Com_Filespec, "NOTRUNNING") ;
  MessageSend(Make_MessageID, "N", ToolClass, "STOP", Com_Filespec) ;
  del_event() ;
  MessageClose();
  Com_Term = 1 ;
  Close() ;
}

del_event() {
  DeleteEvent("R", ToolClass, "START"      , Str_Filespec) ;
  DeleteEvent("R", ToolClass, "STOP"       , Com_Filespec) ;
  DeleteEvent("R", ToolClass, "MAXIMIZE" , Com_Filespec) ;
  DeleteEvent("R", ToolClass, "MINIMIZE" , Com_Filespec) ;
  DeleteEvent("R", ToolClass, "NORMALIZE", Com_Filespec) ;
}
```

Script Example for Evaluating Variable Values

To select the next processing on the basis of the variable values, execute the If.   Use the if for evaluating conditions based on the variable values and for selecting the instruction to be executed according to the logical value.

```
Var data1;
Var data2;
Data1 = input box ("Input data,"   "Input");
If (data1 == "yes")
  {
  Data2 = 200;
    }
    Else
    {
  Data2 = 0;
}
Return;
```

[Note]   "Input box" is a function prepared with script.   See InputBox for details.

Script Example for Selecting a Processing

To select one out of several processing alternatives on the basis of the variable values, execute the Switch.

```
Var data1, data2;
Data1 = input box ("Input data,"   "Input");

Switch (data1)
{
   Case 1:
   Data2 = 100;
   Break;
   Case2:
   Data2 = 200;
   Break;
   Default:
   Data2 = 400;
   Break;
}
Return data2;
```

[Note]   "Input box" is a function prepared with a script.   See InputBox for details.

Script Examples for Repeating Processing

To repeat the processing while a condition is being reached, execute the For, Do, or While. Repeating methods are as follows:

To repeat the processing while waiting for a condition to be reached, use the while and do.

```
Var array 1[10];
Var counter = 0;
While (counter < 10)
{
  Array1[counter] = counter * 100;
  Counter++;
}
Return;
```

[Note] The condition immediately after the while is judged before the block execution. Therefore, the processing in the block may not even be executed once in some cases.

```
Var array1[10];
Var counter = 0;
Do
{
  Array1[counter] = counter * 100;
  Counter++;
}
While (counter < 10);
Return;
```

[Note] The condition immediately after the while is judged after the block execution. Therefore, the block after the do is executed at least once.

To repeat the processing with different variables while a condition is being reached, use the for.

```
Var array1[10];
Var array2[10];
Var counter = 0;
While (counter < 10)
{
  Array1[counter] = counter * 100;
  Counter++;
}
For (counter = 0;
Array1[counter]! = 0;
counter++)
{
  Array2[counter] = array1[counter];
}
Return;
```

[Note] The condition in parentheses (second expression) immediately after the for is judged before the block execution. Therefore, the processing in the block may not even be executed once in some cases.

To stop repeating for a condition, use break.

```
Var array1[10];
Var counter = 0;
While (true)
{
  If (counter >= 9)
  Break;
  Array1[counter] = counter * 10;
  Counter++;
}
Array1[counter] = 0;
Return;
```

To interrupt processing such as repeating and to execute the next processing for a condition, use continue.

```
Var data1 = 0;
Var array2[10];
Var counter = 0;
For (counter = 0;
Counter < 10;)
{
  Data1 = input box ("Input data,"  "Input");
  If (data1 == 0)
  Continue;
  Array2[counter] = data1 * 1000;
  Counter++;
}
Return;
```

[Note] "Input box" is a function prepared with a script. See InputBox for details.

Script Examples for Calling a Function

A script for executing a unit of processing is called a function. An example of calling the function from a script is as follows:

When calling a function without a parameter

Var date data;

Var time data;

Date data = date();

Time data = time();

[Note] "Date" and "time" in the above are functions prepared with a script. See Date and Time for details.

When transferring a parameter without a label

Var data1;

Var data2;

Var data3;

Var data4;

Data1 = 1234;

Data2 = 6789;

Data3 = 4500;

Data4 = average value (data1, data2, and data3);

[Note] "Average value" in the above is the function prepared with a script. See Average for details.

When transferring a parameter with a label

Var data1;

Var data2;

Var data3;

Var data4;

Data1 = 1234;

Data2 = 6789;

Data3 = 4500;

Data4 = function1 (argument3: data3,

argument1: data1,

argument2: data2);

Function1 (argument1: param1, argument 2: param2, argument 3: param3) {

Return (average values (param1, param2, and param3));

}

[Note] "Average value" in the above is a function prepared with script. See Average for details.

Usable Characters

All characters which can be used on the system are usable with a script.   However, the following rules apply:

- Uppercase and lowercase characters except for the specified alphabetic characters in a character string constant are not distinguished.
- Line feed and tab are each considered as a one-character blank.

Usable Numeric Characters

Numeric characters 0 to 9 can be used in script.

How to Write Comments

In script, a character string beginning with "//" and ending with a line feed character is handled as a comment.   However, "//" in a comment and "//" in a character-string constant are exceptions.

Naming Method

Names are unique keywords for identifying messages, variables, and functions.   Names that can be used in a script consist of alphabetic characters, numeric characters, and symbol characters.   A name must always start with an alphabetic character or escape character "("\")"; however, there is no name length limit.

Reserved Words

Reserved words in the script are listed below.   Reserved words have special meanings, and can be written only for special cases.   Therefore, users cannot use a name that is the same as a reserved word.

Reserved words:

```
break
case
continue
default
do
else
false
for
if
null
private
public
return
script
super
switch
true
var
while
```

System Variable

System variables of the script engine are listed below.   These system variables written in format, "Script.system-variable" can be referred to or set.

- ErrorMessage (character-string type)

  This system variable stores the message character string of the last non-interrupting error of the script source being executed.

  This system variable can only be referred to.

- ErrorNumber (numeric type)

  This system variable stores the message number of the last non-interrupting error of the script source being executed.

  This system variable can only be referred to.

- ErrorDisplay (numeric type)

  This system variable indicates whether to display the error messages of non-interrupting errors being executed.

  The error message is displayed if the contents of the system variable is TRUE.   For FALSE, the error message is not displayed.

  This system variable can be referred to or set.

Script Configuration

A script generally consists of the following elements:

- Public variable and private variable definitions
- Function definitions

  Function definitions consist of following elements:

  -- Local variable definitions

  -- Statements

  -- Blocks

A script consists of variable definitions and function definitions. These orders are optional, and each definition is not necessarily grouped in one.

In the function, code the processing for the message received by the object. By coding multiple function definitions, the processing for multiple messages can be coded with one object. If there is only one message received by the object, only the contents of the function definition can be coded.

The script with the function definitions is called Full-set Language Specification, and the script without the function definitions is called Sub-set Language Specification.

A block is a unit of the script enclosed with { and }. Code the group of statements and variable definitions. A block can be nested in a block. A block without contents can also be coded.

When a block is executed, sentences and blocks in the block are executed in sequence. Variables defined in the block are generated when the block is executed and are deleted when the block execution is terminated. The variables defined in the block are effective only in the block.

```
Variable definition A
Statement 1
Statement 2
{                              <-  Block 1 start
  Variable definition B
  Statement 3
  {                            <-  Block 2 start
    Variable definition C
    Statement 4
  }                            <-  Block 2 end
  Statement 5
}                              <-  Block 1 end
Variable definition D
Statement 6
```

Variables that can be referred to in the above sentences (sentence 1 to sentence 6) are as follows:

Sentence 1: A

Sentence 2: A

Sentence 3: A and B

Sentence 4:	A, B, and C
Sentence 5:	A and B
Sentence 6:	A and D

Constants

This section explains the following Constants:

Numeric Constant

A numeric constant indicates a number, and can be coded as either integer or real.

<<Coding>>

- Integer constant

  Numeric list

- Real constant

  Numeric list. numeric list e + numeric list
  Numeric list. numeric list e - numeric list
  Numeric list. numeric list E + numeric list
  Numeric list. numeric list E - numeric list
  Numeric list. numeric list e numeric list
  Numeric list. numeric list E numeric list
  Numeric list. numeric list
  Numeric list. e + numeric list
  Numeric list. e - numeric list
  Numeric list. E + numeric list
  Numeric list. E - numeric list
  Numeric list. e numeric list
  Numeric list. E numeric list
  Numeric list
  Numeric list. e + numeric list
  Numeric list. e - numeric list
  Numeric list. E + numeric list
  Numeric list. E - numeric list Numeric list. e numeric list
  Numeric list. E numeric list
  Numeric list e + numeric list
  Numeric list e - numeric list
  Numeric list E + numeric list
  Numeric list E - numeric list
  Numeric list e numeric list
  Numeric list E numeric list
  Numeric list numeric list

<<Coding examples>>

- To use an integer constant in a script, code the constant as follows:

  12345

- To use a real constant in a script, code the constant as follows:

  123. 45
  123E-3
  123. 45e2
  .45

Character-String Constant

Write the character-string constant with the character list and escape character notation.

<<Coding>>

"Character list"

Usable escape character notations in the character-string constant and their meanings are as follows:

Escape character notation	□F	Meaning
\a	□F	Bell (BL)
\b	□F	Backspace (BS)
\f	□F	Form feed (FF)
\n	□F	New line (line feed) character (NL or LF)
\r	□F	Return (CR)
\t	□F	Horizontal tab (HT)
\v	□F	Vertical tab (VT)
\'	□F	Single quotation (')
\"	□F	Double quotation (")
\\	□F	Yen symbol (\)
\xhh	□F	ASCII character with hexadecimal notation (h: Hexadecimal number, 0 to f)
\ooo	□F	ASCII character with octal notation (o: Octal number, 0 to 7; however, octal numbers, 0 to 3 for the beginning character)

<<Coding example>>

- To use a character-string constant in a script, code the constant as follows:
  "ABC\n"

Logical Constant

The logical constant has a true (non-0) or a false (0) value.   Write "true" for constants having a true value.   Write "false" for constants having a false value.

<u><<Coding>></u>

- To use a logical constant in a script, write the constant with one of the following:

  true

  false

Invalid Constant

Write "null" for the invalid constant indicating the status of having no value stored in the variable.

<<Coding>>

- To use an invalid constant in a script, code the following:

  ```
  null
  ```

Data Types and Type Conversion

Data types

The following five types of data can be used in a script:

Numeric type

Indicates the numeric and logical values (true or false).

Character-string type

Indicates the character string.

Object ID type

Stores the object ID for identifying the object uniquely.

Invalid type

Indicates the status of having no value stored in the variable.   The data type is invalid when only the variable is declared.

Array type

Indicates an array.   An array-type variable is a group of multiple variables, and each variable is called an array element.   Each element is a different type.   If an assignment statement is executed for the array element, the element type is determined.

[Note]   Data type of a variable depends on the value type of the variable.   Therefore, when a variable is defined, the data type of variables other than the variable to which the initial value is set becomes invalid.   However, the array type is determined at variable definition, and the data type of the array element becomes invalid.

Type conversion

For numeric type and character-string type, the data type is converted temporarily depending on the written position according to the following rules:

1) If the character-string type data is coded during arithmetic operation, the character string representing a numeric value is converted to a numeric value and the operation is executed.   If the character-string type data does not represent a numeric value, execution of the script is stopped.
2) At data comparison, character strings are converted to numeric values, and the numeric values are compared as numeric characters in the following cases:
   - When character-string-type data representing numeric values is compared with numeric-type data
   - When character-string-type data representing numeric values is compared with each other
3) At data comparison, numeric values are converted to character strings and the character strings are compared as characters in the following case:
   - When the character-string-type data which does not represent a numeric value is compared with numeric-type data

Data types other than the above are not converted.   For example, if the invalid type was specified for the operand of the arithmetic operation, execution of the script is stopped.

The result of the operation using the arithmetic operator becomes the numeric type.

Variable Definition

Variables are areas for storing data and can be used by defining unique names.

Variables are classified into three types according to the position to be defined:

- Public variable
- Private variable
- Local variable

Public Variable Definitions

A public variable belongs to an object, and can be referred to from other objects.

<<Coding>>

public:

var variable name <, variable name > ...;

var variable name <, variable name = constant-expression > ...;

var variable name <, array > ...;

var variable name = constant-expression <, variable name > ...;

var variable name = constant-expression <, variable name = constant-expression > ...;

var variable name = constant-expression <, array > ...;

var array <, variable name > ...;

var array <, variable name = constant-expression > ...;

var array <, array > ...;

- Array

  Variable name [constant] < [constant] > ...;

[Note] Symbols [and] enclose the expression for specifying the number of array elements and are required.

<<Rules>>

1) Only a variable having a unique name can be specified.
2) An initial value can be specified for the variable (except for an array) with a constant-expression.

<<Coding example>>

- To use a public variable in a script, define the variable as follows:

  public:

  var data1;

  var data2, data3;

With the above definition, defined data1, data2, data3, variable name1, variable name2, and variable name3 can be used in the script. Data1, data2, data3, variable name1, variable name2, and variable name3 are only defined, and their data types become the invalid type.

- To use a public variable having a predefined value in a script, define the variable as follows:

  public:

  var data1 = 10;

  var data2 = "ABC";

With the above definition, 10 is set to defined data1 and variable name1, and "ABC" is set to data2 and variable name2. The data type of data1 and variable name1 becomes the numeric type. The data type of data2 and variable name2 becomes the character-string type.

The data type depends on the set value.

- To use a group of public variables together in a script, define the variables as follows: These are called an array.

  public:

```
    var data1[10];
    var data2[2][3];
```

With the above definition, the data type of data1, data2, variable name1, and variable name2 becomes the array type.   However, the data type of elements (data1[0] to data1[9], data2[0][0] to data2[1][2], variable name1[0] to variable name1[9], and variable name2[0][0] to variable name2[1][2]) is not determined, and becomes invalid.

Private Variable Definitions

A private variable belongs to an object, and cannot be referred to from other objects.

<<Coding>>

 private:

var variable name <, variable name > ...;

var variable name <, variable name = constant-expression > ...;

var variable name <, array > ...;

var variable name = constant-expression <, variable name > ...;

var variable name = constant-expression <, variable name = constant-expression > ...;

var variable name = constant-expression <, array > ...;

var array <, variable name > ...;

var array <, variable name = constant-expression > ...;

var array <, array > ...;

- Array

  Variable name [constant] < [constant] > ...;

[Note]   Symbols [and] enclose the expression for specifying the number of array elements and are required.

<<Rules>>

1) Only a variable having a unique name can be specified.
2) An initial value can be specified for the variable (except for the array) with a constant-expression.

<<Coding example>>

- To use a private variable in a script, define the variable as follows:

  private:

  var data1;

  var data2, data3;

With the above definition, defined data1, data2, data3, variable name1, variable name2, and variable name3 can be used in the script.   Only data1, data2, data3, variable name1, variable name2, and variable name3 are defined, and their data types become invalid.

- To use a public variable having a predefined value in a script, define the variable as follows:

  private:

  var data1 = 10;

  var data2 = "ABC";

With the above definition, 10 is set to defined data1 and variable name1, and "ABC" is set to data2 and variable name2.   The data type of data1 and variable name1 becomes the numeric type.   The data type of data2 and variable name2 becomes the character-string type.

The data type depends on the set value.

- To use a group of private variables together in a script, define the variables as follows: These are called an array.

  public:

```
var data1[10];
var data2[2][3];
```

With the above definition, the data type of data1, data2, variable name1, and variable name2 becomes the array type.   However, the data type of elements (data1[0] to data1[9], data2[0][0] to data2[1][2], variable name1[0] to variable name1[9], and variable name2[0][0] to variable name2[1][2]) is not determined, and becomes invalid.

Local Variable Definitions

Local variables are defined in functions or blocks.

<u><<Coding>></u>

var variable name <, variable name > ...;

var variable name <, variable name = expression > ...;

var variable name <, array > ...;

var variable name = expression <, variable name > ...;

var variable name = expression <, variable name = expression > ...;

var variable name = expression <, array > ...;

var array <, variable name > ...;

var array <, variable name = expression > ...;

var array <, array > ...;

- Array

  Variable name [constant] < [constant] > ...;

[Note]   Symbols [and] enclose the expression for specifying the number of array elements and are required.

<<Rules>>

1) A local variable must be defined before the position to be referred to.
2) Only variables having unique names can be specified in the same block.
3) The initial value can be specified for the variable (except for the array) with the expression.

<<Coding example>>

- To use a variable in a script, define the variable as follows:

  var data1;

  var data2, data3;

With the above definition, defined data1, data2, data3, variable name1, variable name2, and variable name3 can be used in the script.   Only data1, data2, data3, variable name1, variable name2, and variable name3 are defined, and their data types become invalid.

- To use the variable having a predefined value in the script, define the variable as follows:

  var data1 = 10;

  var data2 = "ABC";

With the above definition, 10 is set to defined data1 and variable name1, and "ABC" is set to data2 and variable name2.   The data type of data1 and variable name1 becomes the numeric type.   The data type of data2 and variable name2 becomes the character-string type.

The data type depends on the set value.

- To use a group of variables together in a script, define the variables as follows: These are called an array.

  var data1[10];

  var data2[2][3];

With the above definition, the data type of data1, data2, variable name1, and variable name2 becomes the array type.   However, the data type of elements (data1[0] to data1[9], data2[0][0] to data2[1][2], variable name1[0] to variable name1[9], and variable name2[0][0] to variable name2[1][2]) is not determined, and becomes invalid.

Referring to Variables

Variables can refer to other variables in the same script and variables defined with a script. Variables defined with a script are referred to by object specification reference.

<<Coding>>

Variable name

Array reference

Object specification reference

- Variable name

  Names for identifying variables are called variable names.

  See Naming method for naming regulations of variables.

- Array reference

  Variable name [expression] < [expression]> ...

  Variable name. variable name [expression] < [expression] > ...

  Character-string constant. variable name [expression] < [expression] > ...

  Expression. variable name [expression] < [expression] > ...

[Note] Symbols "[" and "]" enclose the expression for specifying the number of array elements and are required. A "." means array reference in the specified object.

- Object specification reference

  Variable name. variable name

  Character-string constant. variable name

  Expression. variable name

<<Rules>>

1) An array reference expression must be a numeric type or character-string type. It is converted to a numeric type if it is a character-string type.
2) Object specification reference expression must be the object ID type or character-string type.

<<Execution>>

1) If the variable name was specified, the area for the variable name is to be referred to.
2) If an array reference was specified, the element specified with the expression in the array is to be referred to.
3) For object specification reference, searching starts with the specified object and the area of the corresponding variable name is to be referred to.
   1. If a variable or an expression was specified and the variable is a character-string type, the character string is assumed to be the path name, and the specified object is to be referred to. If the variable or the expression is the object ID type, the object specified with the object ID is to be referred to.
   2. If a character-string constant was specified, the character string is assumed to be the path name, and the specified object is to be referred to.
4) For referred-to variables, variables are searched in the following sequence:
   1. Variables in the block
   2. If there are any, variables of the blocks which include the block (continues until no further block includes the block.)
   3. Object variables
   4. Parent object variables

However, for object specification reference, searching starts with the specified object from item 3.

If the variable remains unfound at the end, the execution of the script is stopped.

<<Coding example>>

- To refer to a variable defined in a script, code the variable as follows:

  data1

  Variable name1

- To refer to an array defined in a script, code the following:

  data1[0]

  data2[0][0] ...

  Variable name1 [0]

  Variable name2 [0][0] ...

- To refer to a variable which has been defined in another object, code the following:

  apl1. data1

  "OTHER". data1

  Application1. variable name1

  "Other objects".variable name1

[Note]   apl1 (application1)   must have the ID type.

- To refer to an array which has been defined in another object, code the following:

apl1.data1[0]

Application1.data1[0]

[Notes]

1. apl1 (application1) must have the ID type.
2. Symbols [and] enclose the expression indicating the array elements and are required.

Function Definitions

In the script, a group of processing sequences can be defined as a function. The function definition method is as follows:

<<Coding>>

Function name (parameter declaration)   {processing}

Function name()   {processing}

- Parameter declaration

  Variable name <, variable name > ...

  Variable name = constant-expression <, variable name > ...

  Label: Variable name <, label: variable name > ...

  Label: Variable name = constant-expression <, label: variable name >

[Note] A parameter in the "Label: Variable name" format is called a labeled parameter, and a parameter in other than this format is called a positional parameter.

<<Rules>>

1) Only a variable having a unique name can be specified.
2) The variable name of a parameter declaration is the local variable declared by the outermost block of the function.
3) When "Variable name = constant-expression" is specified in parameter declaration, the value of the constant-expression is set to the variable if the parameter is omitted (or invalid constant is specified)   at calling.
4) A parameter with a labeled and a positional parameter cannot be declared together.

<<Coding examples>>

- To define the function in the script, code the function as follows:

  func1(data1, data2){ ... }

  func2(label1 ; data1,label2 : data2){ ... }

  func3(){ ... }

- To set the default for the variable in a parameter declaration, code the variable as follows:

  func1(data1 = 10, data2 = "ABC"{ ... }

  func2(label1 : data1 = 10, label2 : data2 = "ABC"){ ... }

Calling Functions

In the script, functions can be called for requesting the processing.   The function calling method is as follows:

<<Coding>>

<Variable.> ... function name(parameter list);

<Variable.> ... function name( );

<Character-string constant.> ... function name (parameter list);

<Character-string constant.> ... function name ( );

<Script.> function name(parameter list);

<Script.> function name( );

- Parameter list

  Parameter

  Parameter <, parameter > ...

  Parameter <, parameter > ... <, label;   parameter > ...

  Label: Parameter

  Label: Parameter <, label;   parameter > ...

[Note]   A parameter in the "Label: Parameter" format is called labeled parameter, and a parameter in other than this format is called a positional parameter.

<<Rules>>

1) Variables must be the character-string type or object ID type.
2) Specify the label corresponding to the definition of the function in the calling destination.
3) Even if a labeled parameter has been defined for the function in the calling destination, a parameter can be specified as a positional parameter when specifying the parameter from the beginning.
4) For omitting the positional parameter, specify a comma next to the corresponding position.
5) If multiple parameters have been specified for the same labeled parameter, only the last specified parameter is valid, and other parameters are ignored.
6) If a labeled parameter is specified after the positional parameter, the subsequent parameters must be labeled parameters.
7) For calling a function for which a labeled parameter has not been defined, a labeled parameter must not be specified.
8) If a parameter is omitted, the invalid-type constant is transferred.

<<Execution>>

1) If the function is called in the script, the message is sent for the specified object.
   1. If a character-type variable has been specified, the character-string is assumed to be the path name, and the specified object is to be called.

   If the variable is the object ID type, the object specified with the object ID is to be called.

   2. If a character-string constant has been specified, the character-string is assumed to be the path name, and the specified object is to be called.
   3. If "Script." has been specified, the function prepared by the system is to be called.   For the function prepared by the system, see Script functions.
   4. If none of the above has been specified, the function is searched in the following sequence:

1) Self-object function
2) Parent object function
3) Application in the system

If the function remains unfound at the end, an error occurs.

5. If a variable or a character-string constant has been specified, searching starts with the specified object. In this case, functions defined in the script of the same object are ignored.

2) If the application has been called, the object ID of this application is returned as a return value. Other than that, return values determined for each function are returned.

<<Coding examples>>

- To call a function defined in the script of the same object, code the function as follows:

```
ret1 = func1(data1, data2);
ret2 = func2(label1: data1, label2: data2);
```

- To call a function defined in another object, code the function as follows:

```
ret1 = apl1. func1(data1, data2);
ret2 = "F:\\APL2" . func2(label1: data1, label2: data2);
```

[Note] apl1 must have the object ID type.

- To call a function prepared by the system, code the function as follows:

```
ret = Script . Date();
```

- To specify a labeled parameter and the positional parameter together, code as follows:
  - -- Script for calling the function

    1 which was specified in the positional parameter is replaced by label1: 2 which was specified in the labeled parameter. Therefore, 2 is transferred to para1 of the function in the calling destination.

    ```
    func (1, label1: 2, label2: 3);
    ```

  - -- Definition of the function in the calling destination

```
    func (label1: para1, label2: para2, label3: para3) {
         ...
}
```

Expressions

The following expressions can be used in a script:

Linear expression

Arithmetic expression

Relational expression

Logical expression

The operator used for the expression is explained in the following:

Meanings of operators

Priority of operators

Linear Expression

A linear expression has the following functions:

Referring to variables

Constants

Parentheses expression

Calling functions

When variables are referred to or constants and functions are called, the type of the variable, constant-expression, or calling results are inherited without change.   The value in parentheses is inherited for the parenthetical expression.

Arithmetic Expression

Arithmetic expressions include both unary expressions and dyadic expressions.   A unary expression is an expression with a sign.   A dyadic expression is an expression connected to another expression with an operator.

<u><<Coding>></u>

- Unary expression

  - expression

  + expression

- Dyadic expression

  Expression + expression

  Expression - expression

  Expression * expression

  Expression / expression

<<Rules>>

1) The expression must be a numeric type or character-string type.   For the character-string type, the expression must have a value convertible to the numeric type.
2) The arithmetic result becomes the numeric type.

Relational Expression

Relational expressions include both relational operational expressions and equivalence operational expressions.   A relational operational expression indicates a collating relationship between the left and right parts.   An equivalence operational expression indicates an equivalence relationship between the left and right parts.

<<Coding>>

- Relational operational expression

  Expression < expression

  Expression > expression

  Expression <= expression

  Expression >= expression

- Equivalence operational expression

  Expression == expression

  Expression != expression

<<Rules>>

1) The result of the relational expression becomes the numeric type.   For true, the true (true(1))   value is used.   For false, the false (false(0))   value is used.

2) Combinations between the left and right parts of the relational expression must be as follows:

   - Comparable

     Left part data type: Right part data type

     Numeric type: Numeric type

     Numeric type: Character-string type

     Numeric type: Invalid type (Note: The comparison result always becomes false.)

     Character-string type: Numeric type

     Character-string type;   Character-string type

     Character-string type: Invalid type (Note: The comparison result always becomes false.)

     Invalid type: Numeric type (Note: The comparison result always becomes false.)

     Invalid type: Character-string type (Note: The comparison result always becomes false.)

     Invalid type: Invalid type (Note: The comparison result always becomes true.)

     Invalid type: Object ID type (Note: The comparison result always becomes false.)

     Invalid type: Array type (Note: The comparison result always becomes false.)

     Object ID type: Invalid type (Note: The comparison result always becomes false.)

     Object ID type: Object ID type (Note: The comparison result becomes true if the specified object and the array are equal.)

     Array type: Invalid type (Note: The comparison result always becomes false.)

     Array type: Array type (Note: The comparison result becomes true if the specified object and the array are equal.)

   - Incomparable

     Left part data type: Right part data type

     Numeric type: Object ID type

Numeric type:	Array type
Character-string type:	Object ID type
Character-string type:	Array type
Object ID type:	Numeric type
Object ID type:	Character-string type
Object ID type:	Array type
Array type:	Numeric type
Array type:	Character-string type
Array type:	Object ID type

3) The comparison rules between the numeric type and the character-string type are as follows:

- Numeric comparison

Left part data type:	Right part data type
Numeric type:	Numeric type
Numeric type:	Character-string type (numeric)
Character-string type (numeric):	Numeric type
Character-string type (numeric):	Character-string type (numeric)

- Character comparison

Left part data type:	Right part data type
Numeric type:	Character-string type (non-numeric)
Character-string type (numeric):	Character-string type (non-numeric)
Character-string type (non-numeric):	Numeric type
Character-string type (non-numeric):	Character-string type (non-numeric)

[Note] For numeric comparison, non-numeric type data is converted to numeric and compared. For character comparison, non-character-string-type data is converted to a character string and compared. For type conversion, see Data types and type conversion.

Logical Expression

The logical add, logical product, and negation are obtained with the logical expression.

<u><<Coding>></u>

! Expression

Expression && expression

Expression   expression

<<Rules>>

1) The expression must be the numeric type or character-string type.
2) The result of the logical expression becomes a numeric type.   For true, the true (true(1)) value is used.   For false, the false and (false(0))   value is used.

Meanings of Operators

The meanings of operators are as follows:

- Arithmetic operators

  +(Dyadic operator):   Addition

  -(Dyadic operator):   Subtraction

  *(Dyadic operator):   Multiplication

  /(Dyadic operator):   Division

  +(Unary operator):   No operation

  -(Unary operator):   Inverts the sign.
- Logical operators

  (Dyadic operator):   Logical add

  &&(Dyadic operator):   Logical product

  !(Unary operator):   Negation
- Relational operators

  <(Dyadic operator):   True if the right part is greater than the left part

  >(Dyadic operator):   True if the left part is greater than the right part

  <=(Dyadic operator):   True if the right part is greater than or equal to the left part

  >=(Dyadic operator):   True if the left part is greater than or equal to the right part
- Equivalence operators

  ==(Dyadic operator):   True if the left and right parts are equal

  !=(Dyadic operator):   True if the left and right parts are not equal.
- Assignment operators

  =(Dyadic operator):   Assigns the evaluation result in the right part to the left part.

  +=(Dyadic operator):   Assigns the addition result of evaluation result in the right part and the variable in the left part to the left part.

  -=(Dyadic operator):   Assigns the subtraction result of the evaluation result in the right part and the variable in the left part to the left part

  ++(Unary operator):   Adds 1 to the variable in the left part.

  --(Unary operator):   Subtracts 1 from the variable in the left part.
- Others

  (object specification operator): specifies an object

Priority of Operators

The priority of each operator is as follows:

Priority: Operators

1: ++, --

2: -(Unary), +(Unary), !

3: *, /

4: +(Dyadic), -(Dyadic)

5: <,>,<=,>=

6: ==, !=

7: &&

8: ||

9: -=, +=, =

An object specification operator, array element, and parenthetical expression are evaluated before operators.

Null Statement

The null statement indicates no operation.

<<Coding>>

;

<<Rule>>

The null statement has only a semicolon (;) and can be coded in any position where a statement can appear.

<<Execution>>

No operation.

Assignment Statement

The assignment statement is used for setting or transcribing a value to a variable.

<u><<Coding>></u>

```
Variable = expression;
Variable += expression;
Variable -= expression;
Variable++;
Variable--;
```

<<Execution>>

1) The result of the evaluated expression in the right part is assigned to the left part.
2) With =, the evaluation result of the expression in the right part is assigned to the left part without change.   The type of the expression in the right part is inherited to the type of the variable in the left part without change.
3) With += and -=, the operation result of the right part added to the variable in the left part or the operation result of the right part subtracted from the variable in the left part is assigned to the left part.
4) With ++ and --, 1 is added to the variable in the left part, or 1 is subtracted from the variable in the left part.

<<Example>>

An example of transcribing a value using the assignment statement is listed as follows:

```
var   d1[11] ;
var   d2[11] ;
var   i ;
for (i=0 ; i <= 10 ; i++)
   {
    d1[i] = i;
    d2[i] = i + 10 ;
    }
d1[0] = d2[0] ;
d1[1] += d2[1] ;
d1[2] -= d2[2] ;
d1[10]++ ;
d2[10]-- ;
return;
```

Break

To terminate the processing in the block and transfer control to the next statement, use a break.

<<Coding>>

break;

<<Rule>>

This statement can be written only in the Do, For, Switch, or While.

<<Execution>>

The execution of the do, for, switch, or while including break is terminated. Control is transferred to the next statement of the terminated statement.

<<Example>>

An example of interrupting the loop processing with a condition using break is as follows:

```
var   d1[10] ;
var   d2[11] ;
var   i ;
for (i=0 ; i < 10 ; i++)
   {
     d2[i] = i + 10 ;
     }
i = 0 ;
while(true)
   {
   if (i > 9)
         break ;
   d1[i] = d2[i+1] ;
   i++ ;
   }
d2[i] = null ;
return ;
```

Continue

To skip the processing in the block and execute the processing from the beginning of the next block, use the continue.

<<Coding>>

continue;

<<Execution>>

Left statements of the Do, For, or While including continue are skipped.   Control is transferred to the next repetition of do, for, or while including continue.

<<Example>>

An example of interrupting the processing with a condition using   continue and executing the next loop processing is as follows:

```
var   d1[10] ;
var   d2[10] ;
for ( var i = 0 ;
        i < 10 ;
        i++ )
      {
d1[i] = InputBox("input," "data input," "0");
      if (d1[i] == 0)
           {
           d2[i] = 0 ;
           continue ;
           }
      d2[i] = 100 / d1[i] ;

        }
return ;
```

Do

To repeat the processing while a condition is being reached, use the do.

<<Coding>>

do block-while (expression);

do statement-while (expression);

<<Rule>>

The expression must be a numeric type or character-string type..

<<Execution>>

While the evaluation result of the expression coded after the While is true, the block is executed.   The expression is evaluated after the block execution.   Therefore, the block is executed one or more times.

<<Example>>

An example of repeating the processing while a condition is being reached using do is as follows:

```
var   d1[10] ;
var   i = 0 ;
do
   {
   d1[i] = i * 100 ;
   i++ ;
   }
while( i < 10) G
return ;
```

For

To repeat the processing while a condition is being reached as changing the variable, use the for.

<<Coding>>

```
for (<assignment statement-1>;<logical expression>;<assignment statement-2>)     block
for (<assignment statement-1>;<logical expression>;<assignment statement-2>)
          statement
```

<<Execution>>

1) After the assignment statement-1 is executed, while the logical expression condition is true, the block is executed. If the condition of the logical expression is false, the block is not executed.
2) The assignment statement-2 is executed each time the block is executed.
3) The assignment statement-1, logical expression, and assignment statement-2 can be all omitted.
4) If the logical expression is omitted, the condition is assumed to be true.

<<Example>>

An example of repeating the processing as changing the variable using the for while a condition is being reached is as follows:

```
var   d1[10] ;
var   d2[10] ;
var   i ;
for (i=0 ; i < 10 ; i++)
   {
    d1[i] = i * 10 ;
   }
for (var i=0 ;
       d1[i] != 0 ;
       i++ □j
   {
   d2[i] = d1[i] ;
   }
return ;
```

[Note] The condition in the parentheses (second expression)   immediately after for is judged before the block execution. Therefore, the processing in the block may not even be executed once in some cases.

If

To evaluate the condition based on the variable value and select the instruction according to the logical value, use the if.

<<Coding>>

```
if(expression) block-1 < else block-2 >
if(expression) block-1 < else statement-2 >
if(expression) statement-1 < else block-2 >
if(expression) statement-1 < else statement-2 >
```

<<Rule>>

The expression must be the numeric type or character-string type. The result of the expression is converted to the numeric type and is evaluated as a logical value.

<<Execution>>

1) If the evaluation result of the expression is true (non-0), block-1 or statement-1 after the condition is executed. If the result is false, block-2 or statement-2 is executed.
2) If else has not been specified, the processing is transferred to the next statement of block-1 or statement-1.

<<Example>>

An example of selecting the next processing based on the variable values using if is as follows:

```
var  d1 ;
var  d2 ;
d1 = InputBox("input," "data input," "0");

if (d1 == "yes"□j
  {
    d2=200 ;
  }
else
  {
  d2 = 100 ;
  }
return ;
```

Return

To terminate the processing and return to the calling source, use the return.

<<Coding>>

```
return <expression>;
```

<<Execution>>

1) The function processing is terminated, and control is returned to the sending source or calling source of the window message.
2) If the expression is specified, the value is transferred as the function return value to the sending source or calling source of the window message.
3) The return without specifying the expression is assumed to be written at the end of the function.

Switch

To select one out of several processing alternatives on the basis of the variable values, use the switch.

<<Coding>>

```
switch (expression)   {
case constant-expression:   Statement list
< case constant-expression:   Statement list > ...
< default:   Statement list >
}
```

[Note]   { and } immediately after the expression indicate the range of the block.

<<Execution>>

After the expression is evaluated, the result is compared with the value of the constant-expression continued to each case in sequence.   If the values are the same, the statement is executed.   If no value equal to the case is found, the default is executed if it has been coded.

<<Example>>

An example of selecting one out of several processing alternatives on the basis of the variable values using the switch is as follows:

```
var   d1 ;
var   d2 ;

d1 = InputBox("input," "data input," "0");

switch (d1)
  {
  case   1 :
      d2=100 ;
      break ;
  case   2 :
      d2=200 ;
      break ;
  default :
      d2=0 ;
      break ;
    }
return d2 ;
```

[Note]   If the break is not written, the next statement is executed.

While

<<Coding>>

while (expression)   block

while (expression)   statement

<<Rule>>

The expression must be a numeric type or character-string type.   The expression result is converted to the numeric type and is evaluated as a logical value.

<<Execution>>

1) While the evaluation result of the expression is true (non-0), the block is executed.
2) The expression is evaluated before the block execution.   Therefore, the block may not be executed.

<<Example>>

An example of repeating the processing while a condition is being reached using the while is as follows:

```
var   d1[10] ;
var   d2[10] ;
var   i = 0 ;

while(i < 10)
   {
   d1[i] = d2[i] ;
   i++ ;
   }
return ;
```

Creating and Updating Scripts

Create and update scripts as follows:

- Use a text editor to create or update the script.
  Create or update the script using the PowerFRAMEVIEW editor or the Windows Memo.
  See SAMPLE.SCP for the coding method.

Execution Tools Installed in the PowerFRAMEVIEW

Execute scripts as follows:

- Install the tool in the PowerFRAMEVIEW.

  Install the tool in the PowerFRAMEVIEW Environment setting when the script is created.

- Send the START message to the tool class.

  Send the START Message for the Tool Class of the installed tool.

  When the message is sent, the Macro manager is started, and the script is read and executed.

  Use the development manager, tool manager, or message monitor for sending messages. Refer to the HELP function of each tool for how to use the tool.

the Callback Function

A callback function in a script is called when a message is received.

For calling the callback function, call the MakeEvent and preregister the message pattern.

If the processing for the message could not be executed when the callback function received the message, the message can be requested to be resent with the return value of the callback function.

return (TRUE);: The message is not requested to be resent.

return (FALSE);: The message is requested to be resent.

Other than the above: The message is not requested to be resent.

Exec Function

The exec function in a script is executed immediately after the Macro Manager is started.
Make sure to code the exec function in a script.
The following processing must be coded in this function.

- Opening the message path
  Call the MessageOpen.
- Registering the messages to be receivable in advance
  Call the MakeEvent.

The Terminate Function

The terminate function in a script is executed immediately before the Macro Manager is terminated.

Make sure to code the terminate function in a script.   The terminate function is called by coding Close in a script.

The following processing must be coded in this function.

- Deleting the registered message

  Call the DeleteEvent.
- Closing the message path

  Call the MessageClose.

Debugger Function

This section explains how to use each function in accordance with the debugger status.

- Detecting script errors while checking the contents of variables.
- Changing the execution sequence of a script
- Checking for the changed values of a variable
- Tracing the execution path of a script
- Tracing the calling sequence of a function.

1) Detecting script errors while checking the contents of variables

For checking the validity of the script logic, the control flow and the variable contents can be checked by interrupting the script.   To stop the script execution at a specific position, set the interrupt point for stopping the execution with the execution control function of the debugger main window, and execute the script.

Use the variable display window for checking the variable contents when the script is interrupted.

If a script error is determined by checking the variable contents, change the variable with the variable display window or execute the assignment statement with the script statement execution window.   This corrects the variable contents to the correct value. After the variable is changed to the correct value, debugging of the subsequent script is continued.

2) Changing the execution sequence of a script

A statement containing an error can be skipped or control of the script which has been transferred to the statement in incorrect execution sequence is returned to the statement in the correct sequence with the debugger function.

For changing the script execution sequence, use the 'start point change' command on the execution control menu of the debugger main window.

3) Checking for the changed values of a variable

When the variable value is judged as incorrect, the changed position of the variable value in the script may be unknown.   In such a case, monitor the variable contents, and investigate the cause by checking the value change and interrupting the script when the variable value is changed or a condition is arranged.

Use the variable display window for debugging the script by watching for the variables.

4) Tracing the execution path of a script

To check for the execution path of the script, use the execution trace' command of the execution control menu of the debugger main window.   The script line being executed is reversed on the script display window by enabling the execution trace.   The script execution path can be watched with the reversed display movement.

5) Tracing the calling sequence of a function

The control flow can be checked by checking the window message path sent from a script with the debugger function.

When the script is interrupted, use the function trace window for checking the trace information of the function.

Check for the sequence of sending window messages with the function trace window.

Starting the Debugger

This section explains about starting the debugger.

The debugger is used to detect logical errors in the script while running the script.

The debugger is started in the following conditions:

- If the debug function is executed (when 'debug();' is coded in the script)
- If starting the debugger is specified when the script execution is interrupted
- If the START Message is sent for the debugger installed in the PowerFRAMEVIEW as the DEBUG Tool Class

The debugger indicates the position of the statement being executed or the interrupt position with a reversed character string or marking by displaying the script text being debugged on the screen.   Therefore, control flow can be watched with the debug operation specified directly from the screen.

Refer to the HELP function of the debugger for how to use it.

Encapsulate Functions

The Encapsulate functions for installing tools in the Power FRAMEVIEW are as follows:

- Basic functions

  The function group for opening and closing the message path with the message manager and for registering and sending the message is as follows:

MessageOpen: Opens the message path.

MessageClose: Closes the message path.

MakeEvent: Registers the message pattern to be received.

MessageSend: Sends a message.

Make MessageID: Creates the message ID.

DeleteEvent: Deletes the registered message pattern.

MakeFilespec: Creates the file specification.

FilespecToPathName: Obtains the absolute path name corresponding to the file specification.

- Functions that can be called in the callback function

  The function group that can be called only in the Callback Function called when the message is received is as follows. The information included in the message can be acquired.

GetSenderToolClass: Acquires the class name of the sending source tool of the message.

GetMessageID: Acquires the message ID.

GetMessageType: Acquires the message type.

GetToolClass: Acquires the Tool Class name of the message.

GetCommand: Acquires the command name of the message.

GetFilespec: Acquires the file specification of the message.

GetData: Acquires message data.

GetOriginator: Acquires the Originator of the message.

- Tool operation function

  The function group for operating tools handled in the script is as follows:

ToolExec: Starts the tool (Win16 application).

ToolExec: Starts the tool (Win32 application).

- Script calling function from a script

  The function group for calling another script from a script is as follows:

ScpOpen: Starts the Macro Manager for the Full-set Language Specification, and registers the script written with the full-set language specification.

ScpOpen.: Starts the Macro Manager for the Subset Llanguage Specification

- Macro manager-corresponding function during engine execution

  The function for specifying the correspondence with the Macro Manager during the script engine execution is as follows:

SelfFailureReply: Sets how to correspond to the message when the Macro Manager received a message at script engine execution.

A calling sequence is provided for the Encapsulate function. Call the function in the following sequence:

(1) MessageOpen

(2) MakeFilespec

(3) MakeEvent

(4) FilespecToPathName

(5) GetSenderToolClass

(6) GetMessageID

(7) GetMessageType

(8) GetToolClass

(9) GetCommand

(10) GetFilespec

(11) GetData

(12) GetOriginator

(13) DeleteEvent

(14) MessageClose

(1):   Opens the message path.   This function must be called before calling another Encapsulate function.

(2) to (4):   Registers the message to be received.   These functions can be called anytime if they are called after calling function (1).   Any calling sequence can be applied between functions (2) to (4).

(5) to (12):   Fetches the information of the received message.   Call functions (5) to (12) in the Callback Function.   Any calling sequence can be applied between functions (5) to (12).

(13):   Deletes the registered message.

(14):   Closes the message path.   Other Encapsulate functions must not be called after this function is called.

MessageSend can be called anytime if the message path has been opened.

(1) MessageOpen

(2) MessageSend.

(3) MessageClose

(1):   Opens the message path.   This processing must be executed before calling MessageSend.

(2):   Sends a message.   Messages can be sent anytime.

(3):   Closes the message path.   MessageSend must not be called after this processing is executed.

Functions that can be called anytime regardless of the message path are as follows:

- ToolExec
- ToolExec
- ScpOpen
- ScsOpen
- SetFailureReply

Application Control Function

The application control function for starting other application programs and controlling them remotely is as follows:

Open: Starts an application program.

Screen Control Functions

The screen control functions for controlling the screen are as follows:

FileBox: Displays the dialog box for selecting the file.

InputBox: Displays the dialog box for inputting the character string.

MessageBox: Displays the message box.

SelectBox: Displays the dialog box for selecting the button.

Arithmetic Operation Functions

The arithmetic operation functions for executing the arithmetic operation are as follows:

Abs:	Calculates the absolute value.
Average:	Calculates the average value.
Max:	Reports the maximum value.
Min:	Reports the minimum value.
Mod:	Calculates the remainder.
Power:	Calculates the power.
Rand:	Generates a random number.
Round:	Rounds.
Sqrt:	Calculates the square root.
Srand:	Initializes a random number sequence.
Trunc:	Truncates.

Character String Operation Functions

The functions for character-string operation are as follows:

ByteChange: Changes the character string to another character string (byte specification).

ByteLeft: Fetches the character string from the left-side of the character string (byte specification).

ByteLength: Reports the number of bytes.

ByteMiddle: Fetches the character string from the specified position of the character string (byte specification).

BytePosition: Reports the byte position.

ByteRight: Fetches the character string from the right-side of the character string (byte specification).

ByteSearch: Searches for a character string (byte specification).

Cat: Concatenates the character strings.

Change: Changes the character string to another character string.

CharToCode: Converts the characters to the character code.

CheckNum: Judges whether the character string is usable as a numeric value.

ClassType: Judges the type of character.

CodeToChar: Converts the character code to a character.

CompareString: Compares character strings with each other.

DecToHex: Converts a numeric value to a hexadecimal value.

FormatString: Edits character strings.

HalfToLegal: Converts a half-size character to a full-size character.

HexToDec: Converts a hexadecimal character to a numeric value.

IsAlnum: Judges whether the specified character string is composed of half-size alphanumeric characters.

IsAlpha: Judges whether the specified character string is composed of half-size alphabetic characters.

IsDigit: Judges whether the specified character string is composed of half-size numeric characters.

IsKana: Judges whether the specified character string is composed of half-size katakana characters.

IsLegal: Judges whether the specified character string is full size.

IsLower: Judges whether the specified character string is a half-size lowercase character.

IsUpper: Judges whether the specified character string is a half-size uppercase character.

JisToJms: Converts JIS code to shift JIS code.

JmsToJis: Converts shift JIS code to JIS code.

Left: Fetches the character string from the left side of the character string.

LegalToHalf: Converts full-size characters to half-size characters.

Length: Reports the number of characters.

Middle: Fetches the character string from the specified position of the character string.

MojiPosition: Reports the character position.

NumToString: Converts a numeric value to a corresponding character string.

Right: Fetches the character string from the right-side of the character string.

Search: Searches for a character string.

StringToNum: Converts a character string to a numeric value.

ToLower: Converts an uppercase character in the character string to a lowercase character.

ToUpper: Converts a lowercase character in the character string to an uppercase character.

Date and Time Operation Functions

Functions for reporting the data and time are as follows:

Date: Reports the date (yyyy/mm/dd).

Year: Reports the year (yyyy).

Month: Reports the month (mm).

Day: Reports the date (dd).

DayOfWeek: Reports the day of the week.

Time: Reports the time (hh:mm:ss)

FormatDate: Edits the date.

FormatTime: Edits the time.

ToDate: Obtains the date from the Julian date (yyyy/mm/dd)

ToDays: Obtains the Julian date from the date.

ToSeconds: Obtains seconds from the time.

ToTime: Obtains time from the seconds (hh:mm:ss)

System Functions

The functions for executing the Window system commands are as follows:

Beep:	Generates a beep.
Cd:	Changes the working directory.
CloseTask:	Terminates the program forcibly.
Debug:	Starts the debugger.
Drive:	Changes the working drive.
ExitScript:	Terminates the script.
ExitWindows:	Terminates the Windows system forcibly.
GetObjectID:	Acquires the object ID.
GetProfile:	Reports the Windows environment information.
PostMessage:	Sends various messages.
RefEnv:	References the environment variable.
TaskList:	Displays the task list.
Wait:	Waits for the task.
WindowsInformation:	Reports the Windows information.

Type Checking Functions

The functions for reporting variable types are as follows:

IsArray: Judges the array type.

IsNull: Judges the null type.

IsNum: Judges the numeric type.

IsObj: Judges the object ID type.

IsString: Judges the character-string type.

VarType: Reports the type of the expression.

DDE Conversation Functions

The DDE conversation functions are as follows:

DDEInitiate: Starts the DDE conversation.

Execute: Requests the command execution.

Poke: Sends data.

Request: Requests data.

Terminate: Terminates DDE conversation.

Advise: Requests the server to open the hot link.

Unadvise: Terminates items opened with the hot link to the server.

DLL Functions

The functions for calling the DLL are as follows:

DLLDeclare: Defines the DLL calling function.

DLLDelete: Deletes the DLL calling library.

DLLLoad: Loads the DLL calling library.

DLLcalling: Calls the DLL calling function.

Timer Functions

The functions for operating the timer are as follows:

Timer: Interrupts the processing up to the specified time.

Sleep: Interrupts processing for the specified number of seconds.

Array Operation Functions

The functions for operating the array are as follows:

CreateArray: Creates the array.

DuplicateArray: Duplicates the array.

NumberArray: Reports the number of array items.

Window Operation Function

The function for the operating window is as follows:

Close: Terminates the Macro Manager processing.

Dialog Box Functions

The functions for displaying dialog boxes are as follows:

CreateDialogBox: Creates a dialog box.

AddCheckBox: Adds a check box.

AddComboBox: Adds a combo box.

AddEditText: Adds edit control.

AddGroupBox: Adds a group box.

AddListBox: Adds a list box.

AddPushButton: Adds a push button.

AddRadioButton: Adds a radio button.

AddText: Adds text control.

Show: Displays a dialog box.

Delete: Deletes a dialog box.

Expanded Dialog Box Functions

This section explains the Extended Dialog Box Function.

The basic function is the same as the dialog box function; however, some functions have been added or changed.

Some functions have also been added in accordance with addition or change of the original functions.

Added or changed functions are as follows:

- Displaying the modeless dialog box

  The Modal Dialog Box is displayed in the dialog box function.   The Modeless Dialog Box is displayed in the extended dialog box function.
- Specifying the style

  Each control style can be specified (except for a radio button and a group box).
- Dynamic dialog box

  The dialog box is not closed by clicking the push button.   The dialog box remains displayed until the CloseDialogBox function is called.
- Adding and Deleting an item

  A character string can be added or deleted in the list box.
- Selecting multiple items

  Multiple character strings in the list box can be selected.
- Releasing the item selection status

  The selection status of the list box can be released.
- Displaying a horizontal scroll bar

  A horizontal scroll bar can be displayed in the list box.
- Displaying the directory name

  The directory name can be displayed in the list box.
- Displaying the drive name

  The drive name can be displayed in the combo box.
- Managing the pushbutton status

  The pushbutton can be put into the selectable status or nonselectable status.
- Adding new functions:

  CloseDialogBox

  ExistDialogBox

  AddListString

  DeleteListString

  EnablePushButton

  DisablePushButton

  GetPushButton

  ResetPushButton

  ClearListSelect

The following function group cannot be used together with the dialog box functions.

For example, control cannot be added to the extended dialog box object acquired with the extended dialog box function by calling the dialog box function.

CreateDialogBoxEx:	Creates an extended dialog box.
AddTextEx:	Adds extended text control.
AddCheckBoxEx:	Adds an extended check box.
AddPushButtonEx:	Adds an extended push button.
AddRadioButtonEx:	Adds an extended radio button.
AddEditTextEx:	Adds extended edit control.
AddGroupBoxEx:	Adds an extended group box.
ShowEx:	Displays an extended dialog box.
DeleteEx:	Deletes an extended dialog box.
AddListBoxEx:	Adds an extended list box.
AddComboBoxEx:	Adds an extended combo box.
CloseDialogBox:	Closes an extended dialog box.
ExistDialogBox:	Checks display of the extended dialog box.
AddListString:	Adds an item to the list box.
DeleteListString:	Deletes an item from the list box.
EnablePushButton:	Puts the pushbutton into the selectable status.
DisablePushButton:	Puts the pushbutton into the nonselectable status.
GetPushButton:	Reports the pushbutton being selected.
ResetPushButton:	Releases the pushbutton being selected.
ClearListSelect:	Releases the item selected in the list box.

File Operation Functions

The functions for operating files are as follows:

OpenFile: Opens a file.

Close: Closes a file.

Read: Reads data from the file.

Write: Writes data to the file.

Eof: Reports the end of a file.

FileList: Reports the file names.

CopyFile: Copies a file.

DeleteFile: Deletes a file.

RenameFile: Changes the file name.

MakeDirectory: Creates a sub-directory.

RemoveDirectory: Deletes a sub-directory.

RemoveDirectoryAll: Deletes a sub-directory and files and sub-directories in the sub-directory.

ExistFile: Reports file existence.

FileAttribute: Reports the file attribute.

GetDrive: Reports the drive name.

GetPath: Reports the path name.

GetFileName: Reports the file name.

GetName: Reports the name of a file.

GetExtension: Reports the extension of the file.

SetAttribute: Sets the file attribute.

SuitableName: Generates a nonexisting file name

Other Functions for Operating the Window

RestoreWindow: Restores the window of other tools.

MoveWindow: Moves the window of other tools.

SizeWindow: Changes the window size of other tools.

MaximizeWindow: Maximizes the window size of other tools.

MinimizeWindow: Minimizes the window size of other tools.

CloseWindow: Closes the window of other tools.

ActivateWindow: Activates the window of other tools.

GetWindowPos: Acquires the window position of other tools.

GetWindowSize: Acquires the window size of other tools.

BottomWindow: Locates the window of other tools at the bottom of the screen.

Clipboard Operation Functions

The functions for operating the clipboard are as follows:

ClipboardCopy: Copies data to the clipboard.

ClipboardPaste: Reads data from the clipboard.

Bit Operation Functions

The functions for executing bit operation are as follows:

BitAnd: Obtains the logical product.

BitOr: Obtains the logical sum.

BitXor: Obtains the exclusive logical sum.

BitNegate: Reverses the bit.

BitShiftLeft: Shifts the arithmetic operation to the left.

BitShiftRight: Shifts the arithmetic operation to the right.

List Processing Functions

The functions for executing the List processing are as follows:

Car: Fetches the head Item of the list.

Cdr: Deletes the head item of the list.

Element: Fetches the item in the specified position from the list.

ElementPos: Obtains the position of the item in the list.

GetElementsAll: Fetches all items of the list.

List: Creates the list.

NumElements: Obtains the number of items in the list.

Script Evaluation Function

The function for evaluating the script is as follows:

Eval: Executes the specified script.

Parameter Operation Function

The function for operating the parameter is as follows:

Getparameter: Receives the parameter specified at script execution.

Event Functions [Win16]

The functions for generating various simulated operations are as follows:

KeyBoard: Generates simulated input from the keyboard.

Menu: Generates simulated menu operation.

Mouse: Generates simulated mouse operation.

Scrollbar: Generates simulated scroll bar operation.

Win 16 Application Control Functions [Win32]

The functions for controlling the Win16 application are as follows:

Open16: Starts the Win16 application.

Wait16: Waits for completion of the Win16 application.

Open

<<Function>>   Opens the specified file.   If the extension of the specified file is EXE, the application specified with the file name is started.   With non-EXE extensions, related applications are started, and the file specified with the file name is opened.

The parameter specified with the parameter character string is transferred to the started application.

<<Coding>>   File-Name.Open(<parameter><,parameter> ...)

In File-Name, specify the name of the file to be opened with a character string.

<<Argument>>   Parameter = Specify the parameter to be transferred to the application specified with File-Name using up to 18 characters.

<<Return Value>>   If application start is successful, the object ID indicating the application is reported.

<<Win32>>

Use the definition of the extension defined in the registry.   If this function was used to start the Win16 application, the correct object ID cannot be acquired.

- Call Open16 for starting the Win16 application.

<<Example>>   var application ID;

ApplicationID = "FILE01.EXE".Open(); <- 1

ApplicationID = "DATA01.XJS".Open(); <- 2

If 1 is executed, File01.EXE is started, and the object ID indicating File01.EXE is set for variable application ID.

If 2 is executed, the DATA01.XJS is opened after Excel is started when the DATA01.XJS is related to "MS-Excel."

FileBox

<<Function>>   Displays files in the dialog box, and reports the selected file names.

<<Coding>>   FileBox(title<,<file name><,<type><,<mode>>>>)

<<Argument>>

Title = Specify the title of the dialog box.

File name = Specify the initial value of the file name to be displayed in the dialog box.   No value is displayed if omitted.

Type = Specify the wild card of the file name to be displayed in the file list.   The default is *.*.

Mode = Specify the mode of the file to be listed with SAVE (for creation) or OPEN (for update).   Whether the selected file exists is checked according to the specified mode, and the confirmation message is displayed in the cases shown below.   If the mode is omitted, whether the file exists is not checked.

1) If SAVE is specified and the filed matched with the selected file already exists:

   "The specified file already exists.   Change?"

   ->   Yes or No

   For changing, specify "Yes," and for reselecting a file, specify "No."

2) If OPEN is specified and the file matched with the selected file does not exist:

   "The specified file does not exist."

   -> "OK"

   Returned from this function only when the existing file is selected or the specification is canceled in this case.

<<Return Value>>   Selecting the OK button will report the selected file name.   Selecting the cancel button will report null.

InputBox

<<Function>>   Displays the dialog box, and reports the input character strings.

<<Coding>>   InputBox(message, title<,initial value>)

<<Argument>>

Message = Specify the message to be displayed in the dialog box.

Title = Specify the title of the dialog box.

Initial value = Specify the initial value of the input character string.   The default is blank.

<<Return Value>>   Selecting the OK button will report the input character string.
Selecting the cancel button will report null.

<<Example>>   var data;

Data = InputBox("Input data.","input," "initial value")

Selecting the OK button in the above dialog box sets character string "initial value" to variable "data."   Selecting the cancel button sets null to variable "data."

MessageBox

<<Function>>   Displays the message box, and waits for the reply for the message.

<<Coding>>   MessageBox(message,title<,style>)

<<Argument>>

Message = Specify the message to be displayed in the message box.

Title = Specify the title to be displayed in the message box.

Style = Specify the style of the message box (display formats of buttons and the icons). Use 0 to 5 to specify the display format of buttons.   Specify the value added to one of 16, 32, 48, or 64 for displaying the icon.   The default is the standard style (0).   Styles of the message box corresponding to parameters are as follows:

Display format	Style
0	OK button only
1	OK and cancel buttons
2	Stop, retry, and ignore buttons
3	Yes, no, and cancel buttons
4	Yes and no buttons
5	Retry and cancel buttons
16	STOP icon
32	? icon
48	! icon
64	! icon

The button text (such as OK or cancel) is coded using Windows V3.1 as an example.   The text on buttons may differ depending on the Windows system version.

<<Return Value>>   Values corresponding to the buttons selected in the message box are reported.   Otherwise, null is reported.   The relationship between the buttons and the return values are as follows:

Button	Return value
OK	1
Cancel	2
Stop	3
Retry	4
Ignore	5
Yes	6
No	7

SelectBox

<<Function>>   Displays the dialog box, and reports the character string corresponding to the selected button.

<<Coding>>   SelectBox(message, title,button1<,button2>...)

<<Argument>>

Message = Specify the message to be displayed in the dialog box.

Title = Specify the title to be displayed in the dialog box.

Button = Specify the name for the push button selectable in the dialog box.   The push button with the name specified for button1 becomes the default push button.

<<Return Value>>   The character string of the selected push button is reported.

<<Example>>   var selected result;

Selected result = SelectBox("Select the button being displayed."

"selection," "button1," "button2," or "button3");

Selecting "Button1" in the above dialog box sets character string "button1" to variable "selection result."

Abs

<<Function>>   Obtains the absolute value of the specified value.

<<Coding>>   Abs(numeric value)

<<Argument>>   Numeric value = Specify the value of which the absolute value is to be obtained.

<<Return Value>>   The absolute value of the value specified with the argument is reported.

Average

<<Function>>   Obtains the average value of the specified values.

<<Coding>>   Average(numeric1 <,numeric2>...)

<<Argument>>   Numeric n = Specify the values of which the average value is to be obtained.

<<Return Value>>   The average value of the values specified with the argument is reported.

Max

<<Function>>   Obtains the maximum value of the specified values.

<<Coding>>   Max(numeric1 <,numeric2>...)

<<Argument>>   Numeric n = Specify the values of which maximum value is to be obtained.

<<Return Value>>   The maximum value of the values specified with the argument is reported.

Min

<<Function>>   Obtains the minimum value of the specified values.

<<Coding>>   Min(numeric1 <,numeric2>...)

<<Argument>>   Numeric n = Specify the values for which the minimum value is to be obtained.

<<Return Value>>   The minimum value of the values specified with the argument is reported.

Mod

<<Function>>   Obtains the remainder of the specified division.

<<Coding>>   Mod(dividend,divisor)

<<Argument>>

Dividend = Specify the dividend.

Divisor = Specify the divisor.

<<Return Value>>   The remainder of the division specified with the argument is reported.

Power

<<Function>>   Obtains the power of the specified value.

<<Coding>>   Power(base,exponent)

<<Argument>>

Base = Specify the base of the power.

Multiplier = Specify the exponent of the power.

<<Return Value>>   The power of the value specified with the argument is reported.

Rand

<<Function>>   Obtains a random number.

<<Coding>>   Rand()

<<Argument>>   None

<<Return Value>>   Random number (0 or more but less than 1) is reported.

Round

<<Function>>   Obtains the result of rounding a specified value to the specified digit position.

<<Coding>>   Round(numeric value<,digit position>)

<<Argument>>

Numeric value = Specify the value to be rounded off.

Digit position =   Specify the digit position of the rounded value.   If a positive value is specified, the fraction part is rounded.   If 0 or a negative value is specified, the integer part is rounded.   The default is 0 (rounded off to the first decimal place).

<<Return Value>>   The rounded value specified with the argument is reported.

<<Example>>   var result;

Result = Round(910.2345,3);

The above example rounds off the fourth decimal place 5 and sets 910.235 to variable "result."

Result = Round(12345.67,-1);

The above example rounds the 5 at the ones position of the integer part and sets 12350 to variable "result."

Result = Round(-1.5)

The above example rounds off the first decimal place to 5 and sets -2 to variable "result."

Sqrt

<<Function>>   Obtains the square root of the specified value.

<<Coding>>   Sqrt(numeric value)

<<Argument>>   Numeric value = Specify the value for which the square root is to be obtained.

<<Return Value>>   The square root of the value specified with the argument is reported.

Trunc

<<Function>>   Obtains the result of truncating a specified value to the specified position.

<<Coding>>   Trunc(numeric value<,digit position>))

<<Argument>>

Numeric value = Specify the value to be truncated.

Digit position =   Specify the digit position of the truncated value.   If a positive value is specified, the fraction part is truncated.   If 0 or a negative value is specified, the integer part is truncated.   The default is 0 (first decimal place is truncated).

<<Return Value>>   The truncated value of the value specified with the argument is reported.

<<Example>>   var result;

Result = Trunc(910.2345,3);

The above example truncates fourth decimal place 5 and sets 910.234 to variable "result."

Result = Trunc(12345,-3);

The above example truncates up to the 3 of the hundreds position of the integer part and sets 12000 to variable "result."

Srand

<<Function>>   Initializes the specified random number sequence.

<<Coding>>   Srand (seed)

<<Argument>>   Seed = Specify the value for generating the random number sequence.
If a value outside the range of 0 to 65535 is specified, 0 is assumed to be specified.

<<Return Value>>   None.

ByteLength

<<Function>> Reports the number of bytes of the specified character string.

<<Coding>> ByteLength(character string)

<<Argument>> Character string = Specify the character string of which number of bytes is to be obtained.

<<Return Value>> The number of bytes is reported.

Cat

<<Function>>   Concatenates the specified character strings.

<<Coding>>   Cat(character string1<,character string2>...)

<<Argument>>   Character string n = Specify the character string to be concatenated.

<<Return Value>>   The concatenated character strings are reported.

Change

<<Function>>   Searches for an optional character string and changes the character string to the specified character string.   If the character string to be changed exceeds the end position of changing, the character string is not changed.

<<Coding>>   Change(character string1, character string2, character string3<,start position><,end position>)

<<Argument>>

Character string1 = Specify an optional character string.

Character string2 = Specify the character string to be changed.

Character string3 = Specify a new character string.

Start position = Specify the start position of changing.   If omitted or if a negative number is specified, 0 is assumed.

End position = Specify the end position of changing.   If omitted or if the character string length is longer than the length of the optional character string, the end of the character string is assumed to be specified.

<<Return Value>>   The changed character string is reported.

CharToCode:

<<Function>>   Reports the character code of the first character (full-size or half-size) of the specified character string.

<<Coding>>   CharToCode(character string)

<<Argument>>   Character string = Specify the character string of which the character code is to be known.

<<Return Value>>   The following codes according to the specified character are reported:

Half-size character: 1 to 128, 160 to 223, or 253 to 255

Full-size character: 33024 to 40959 or 57344 to 64767

If the length of the specified character string is 0, 0 is reported.

CodeToChar

<<Function>>   Reports the character corresponding to the specified character code.   For half-size characters, specify 1 to 128, 160 to 223, or 253 to 255.   For full-size characters, specify 33024 to 40959 or 57344 to 64767.

<<Coding>>   CodeToChar(character code)

<<Argument>>   Character code = Specify the character code to be converted.

<<Return Value>>   The corresponding character is reported.

StringToNum

<<Function>>   Converts the specified character string to a numeric value.   Converting rules conform to the script language specification.

<<Coding>>   StringToNum(character string)

<<Argument>>   Character string = Specify the character string to be converted.

<<Return Value>>   The converted numeric value is reported.

NumToString

<<Function>>   Converts the specified numeric value to a numeric character string.

<<Coding>>   NumToString(numeric value)

<<Argument>>   Numeric value = Specify the numeric value to be converted to a character string.

<<Return Value>>   The converted character string is reported.

ToUpper

<<Function>>   Converts the half-size lowercase characters in the specified character string to half-size uppercase characters.

<<Coding>>   ToUpper(character string)

<<Argument>>   Character string = Specify the character string to be converted.

<<Return Value>>   The converted character string is reported.

ToLower

<<Function>>   Converts the half-size uppercase characters in the specified character string to half-size lowercase characters.

<<Coding>>   ToLower(character string)

<<Argument>>   Character string = Specify the character string to be converted.

<<Return Value>>   The converted character string is reported.

Left

<<Function>>   Fetches the specified number of characters from the left side of the specified character string.

<<Coding>>   Left(character string,number of characters)

<<Argument>>

Character string = Specify the character string source.

Number of characters = Specify the number of characters to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.

If the specified number of characters is longer than the character string length specified with the character string, the character string specified with the character string is reported without change.

Middle

<<Function>>   Fetches the specified number of characters starting from the specified fetching position of the specified character string and going to the left.

<<Coding>>   Middle(character string, fetching position,number of characters)

<<Argument>>

Character string = Specify the character string source.

Fetching position = Specify the character fetching position with 0-relative.

Number of characters = Specify the number of characters to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.

If the specified number of characters is less than the character string length specified, up to the end of the character string specified with the character string is reported.   If the fetching position is greater than the character string length, null is reported.

Right

<<Function>>   Fetches the specified number of characters from the right side of the specified character string.

<<Coding>>   Right(character string,number of characters)

<<Argument>>

Character string = Specify the character string source.

Number of characters = Specify the number of characters to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.

If the specified number of characters is greater than the character string length specified with the character string, the character string specified with the character string is reported without change.

Search

<<Function>>   Searches for the specified search character string among the specified character strings.

<<Coding>>   Search(character string, search character string<,searching position>)

<<Argument>>

Character string = Specify the character string to be searched.

Search character string = Specify the search character string.

Searching position = Specify the searching start position with 0-relative.   The default is 0.

<<Return Value>>   The position of which the search character string is found is reported. If not found, -1 is reported.   If the length of the character string or the search character string is 0, -1 is reported.   If the character string length is shorter than the length of the search character string, 0 is reported.   If the searched position is less than 0, -1 is reported.

Length

<<Function>>   Reports the number of characters of the specified character string.

<<Coding>>   Length(character string)

<<Argument>>   Character string = Specify the character string for which the number of characters is to be obtained.

<<Return Value>>   Number of characters is reported.

CheckNum

<<Function>>   Reports whether the specified character string can be used as a numeric value.

<<Coding>>   CheckNum(character string)

<<Argument>>   Character string = Specify the character string to be judged.

<<Return Value>>   If the character string can be used as a numeric value, true is returned. Otherwise, false is returned.

FormatString

<<Function>>   Edits the specified character strings and numeric values.

<<Coding>>   FormatString(format,string)

<<Argument>>

Format = Specify the conversion directive and the conversion character with a character string started with %.   If the conversion directive is to be specified, specify it before the conversion character.   Specifiable conversion directives and characters are as follows:

- Conversion directives

-	:	Outputs left decimal alignment.
+	:	Puts a sign for positive numbers.
0	:	Does not output blanks for leading 0s.
dd	:	Number of digits of entire field (dd: positive decimal number)
Dd	:	Splits the field, and makes the precision digit in the field to dd (positive decimal number)

- Conversion characters

Cd	:	Decimal number with a comma for every three digits
Cf	:	Number of floating points in the ddd.ddd format with a comma for every three digits
d	:	Decimal number
f	:	Number of floating points in the ddd.ddd format
e	:	Number of floating points in the d.ddde+ddd format
s	:	Character string

String = Specify the character string or a numeric value to be edited

<<Return Value>>   The edited character strings are reported.

<<Example>>   var edit result;

Edit result = FormatString("%+11.2Cf",12345.67)

The above example reports character string "+123.345.67" as the edit result.

CompareString

<<Function>>   Compares the specified character strings.

<<Coding>>   CompareString(character string1,character string2)

<<Argument>>

Character string1 = Specify the character string to be compared.

Character string2 = Specify the character string to compare.

<<Return Value>>   The following values are reported according to the compared result:

If matched, 0 is reported.

If character string1 is greater than character string2, a positive value is reported.

If character string1 is less than character string2, a negative value is reported.

ByteChange

<<Function>>   Searches for an optional character string and changes it to the specified character string.   If the character string to be changed exceeds the end position of changing, it is not changed.   Specify the start and end positions of changing with the number of bytes from the beginning.

<<Coding>>   ByteChange(character string1, character string2, character string3<,start position><,end position>)

<<Argument>>

Character string1 = Specify an optional character string.

Character string2 = Specify the character string to be changed.

Character string3 = Specify a new character string.

Start position = Specify the start position of changing.   If omitted or if a negative number is specified, 0 is assumed.

End position = Specify the end position of changing.   If omitted or if the end position is longer than the length of the optional character string, the end of the character string is assumed.

<<Return Value>>   The changed character string is reported.   If changing starts from the second byte of a double-byte character code, changing starts from the next character after the second byte.   If changing ends at the first byte of a double-byte character code, characters are changed up to the previous character.

ByteLeft

<<Function>>   Fetches the specified number of bytes from the left side of the specified character string.

<<Coding>>   ByteLeft(character string,number of bytes)

<<Argument>>

Character string = Specify the character string source.

Number of bytes = Specify the number of bytes to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.   If the specified number of bytes is longer than the character string length specified with the character string, the character string specified with the character string is reported without change.   If fetching ends at the first byte of a double-byte character code, characters up to the previous character are reported.

ByteMiddle

<<Function>>   Fetches the character string for the specified number of bytes starting from the specified fetching position of the specified character string and going to the left.

<<Coding>>   ByteMiddle(character string, fetching position,number of bytes)

<<Argument>>

Character string = Specify the character string source.

Fetching position = Specify the character fetching position with the number of bytes (0-relative).

Number of bytes = Specify the number of bytes to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.   If the specified character string length is shorter than the character string length specified with the character string, up to the end of the character string specified with the character string is reported.   If the fetching position is longer than the character string length, null is reported.

If fetching starts at the second byte of a double-byte character code, characters from the next character to the second byte are reported.   If fetching ends at the first byte of a double-byte character code, characters up to the previous character are reported.

BytePosition

<<Function>>   Reports the byte position of the character specified in the character string with the number of bytes from the beginning.

<<Coding>>   BytePosition(character string,character position)

<<Argument>>

Character string = Specify the character string of which byte position is to be known.

Character position = Specify the character position (number of characters from the beginning) with 0-relative.

<<Return Value>>   The byte position (0-relative) of the specified character is reported.

If null is specified to the character string, -1 is reported.   If the character position is longer than the character string length, null is reported.

ByteRight

<<Function>>   Fetches the specified number of bytes from the right side of the specified character string.

<<Coding>>   ByteRight(character string,number of bytes)

<<Argument>>

Character string = Specify the character string source.

Number of bytes = Specify the number of bytes to be fetched.

<<Return Value>>   If fetching is successful, the fetched character string is reported.   If the specified number of bytes is longer than the character string length specified with the character string, the character string specified with the character string is reported without change.   If fetching starts with the second byte of a double-byte character code, the specified character string is reported, including the first byte of the double-byte character code at the fetching start position.

ByteSearch

<<Function>>   Searches for the specified searching character string among the specified character strings.

Specify the searching character position with the number of bytes from the beginning.   The position at which the search character string is found is reported with the number of bytes.

<<Coding>>   ByteSearch(character string, search character string<,searching position>)

<<Argument>>

Character string = Specify the character string to be searched.

Search character string = Specify the search character string.

Searching position = Specify the searching start position with 0-relative.   The default is 0.

<<Return Value>>   The position of which search character string is found is reported (number of bytes).   If not found, -1 is reported.   If the length of the character string or the search character string is 0, -1 is reported.   If the character string length is shorter than the length of the search character string, 0 is reported.   If the searched position is less than 0, -1 is reported.

ClassType

<<Function>>   Judges the character type of the specified character string.

<<Coding>>   ClassType(character string)

<<Argument>>   Character string = Specify the character string to be judged.

<<Return Value>>   Character strings indicating the judged type are reported as follows:

Character string	:	Meaning
ALPHA	:	All half-size alphabetic characters
LOWER	:	All half-size lowercase characters
UPPER	:	All half-size uppercase characters
DIGIT	:	All half-size numeric characters
ALNUM	:	All half-size alphanumeric characters
KANA	:	All half-size katakana characters
HALF	:	All half-size characters (including symbols)
LEGAL	:	All full-size characters
MIX	:	Mixed-size characters (full-size and half-size)
NSCP	:	All null characters
OTHER	:	Characters other than the above

DecToHex

<<Function>>   Converts the specified numeric value to a character string with hexadecimal notation

<<Coding>>   DecToHex(numeric)

<<Argument>>   Numeric value = Specify the numeric value to be converted to a character string with hexadecimal notation.

<<Return Value>>   The character string with hexadecimal notation is reported.

The hexadecimal notation of the converted character string is four-byte notation.

<<Example>>   Examples of conversion results are as follows:

Numeric:	Conversion result
125:	"7D"
-50:	"FFFFFFCE"

HalfToLegal

<<Function>>   Converts a half-size character in the specified character string to a full-size character.

<<Coding>>   HalfToLegal(character string)

<<Argument>>   Character string = Specify the character string in which a half-size character is to be converted to a full-size character.

<<Return Value>>   The converted character string is reported.

Katakana and alphanumeric characters can be converted to full-size characters.

If characters which cannot be converted to full-size characters are specified in the character string or null is specified, the specified characters are reported without change.

HexToDec

<<Function>>   Converts the specified character string with hexadecimal notation to a numeric value.

<<Coding>>   HexToDec(character string)

<<Argument>>   Character string = Specify the character string with hexadecimal notation to be converted to a numeric value.

<<Return Value>>   The converted numeric value is reported.

The hexadecimal notation of the character string to be specified is four-byte notation.

<<Example>>   Examples of conversion results are as follows:

Character string:	Conversion result
7D or 0x7D:	125
FFFFFFCE or 0xFFFFFFCE:	-50

IsAlnum

<<Function>>　Judges whether the specified character string is composed of half-size alphanumeric characters.

<<Coding>>　IsAlnum(character string)

<<Argument>>　Character string = Specify whether the character string to be judged is composed of half-size alphanumeric characters.

<<Return Value>>　If all of the judged character string is composed of half-size alphanumeric characters, true is reported.　Otherwise, false is reported.

IsAlpha

<<Function>>   Judges whether the specified character string is composed of half-size alphabetic characters.

<<Coding>>   IsAlpha(character string)

<<Argument>>   Character string = Specify the character string to be judged whether it is composed of half-size alphabetic characters.

<<Return Value>>   If all of the judged character string is composed of half-size alphabetic characters, true is reported.   Otherwise, false is reported.

IsDigit

<<Function>>　Judges whether the specified character string is composed of half-size numeric characters.

<<Coding>>　IsDigit(character string)

<<Argument>>　Character string = Specify the character string to be judged whether it is composed of half-size numeric characters.

<<Return Value>>　If all of the judged character string is composed of half-size numeric characters, true is reported.　Otherwise, false is reported.

IsKana

<<Function>>　Judges whether the specified character string is composed of half-size katakana characters.

<<Coding>>　IsKana(character string)

<<Argument>>　Character string = Specify whether the character string to be judged is composed of half-size katakana characters.

<<Return Value>>　If all of the judged character string is composed of half-size katakana characters, true is reported.　Otherwise, false is reported.

IsLegal

<<Function>>　Judges whether the specified character string is full-size.

<<Coding>>　IsLegal(character string)

<<Argument>>　Character string = Specify whether the character string to be judged is full-size.

<<Return Value>>　If the judged character string is full-size, true is reported.　Otherwise, false is reported.

IsLower

<<Function>>　Judges whether the specified character string is composed of half-size lowercase characters.

<<Coding>>　IsLower(character string)

<<Argument>>　Character string = Specify whether the character string to be judged is composed of half-size lowercase characters.

<<Return Value>>　If all of the judged character string is composed of half-size lowercase characters, true is reported.　Otherwise, false is reported.

IsUpper

<<Function>>　Judges whether the specified character string is composed of half-size uppercase characters.

<<Coding>>　IsUpper(character string)

<<Argument>>　Character string = Specify whether the character string to be judged is composed of half-size uppercase characters.

<<Return Value>>　If all of the judged character string is composed of half-size uppercase characters, true is reported.　Otherwise, false is reported.

JisToJms

<<Function>>   Converts JIS code in the specified character string to shift JIS code.

<<Coding>>   JisToJms(character string)

<<Argument>>   Character string = Specify the character string to be converted to shift JIS code.

<<Return Value>>   The character string converted to shift JIS code is reported.

If characters which cannot be converted to shift JIS code are specified or null is specified, the specified characters are reported without change.

If the specified character string contains 1-byte and double-byte characters, the conversion result is not guaranteed.

JmsToJis

<<Function>>   Converts shift JIS code in the specified character string to JIS code.

<<Coding>>   JmsToJis(character string)

<<Argument>>   Character string = Specify the character string to be converted to JIS code.

<<Return Value>>   The character string converted to JIS code is reported.

If characters which cannot be converted to the JIS code are specified or null is specified, the specified characters are reported without change.

LegalToHalf

<<Function>>   Converts full-size characters in the specified character string to half-size characters.

<<Coding>>   LegalToHalf(character string)

<<Argument>>   Character string = Specify the character string of which full-size characters are to be converted to half-size characters.

<<Return Value>>   The converted character string is reported.   Katakana and alphanumeric characters can be converted to half-size characters.

If characters which cannot be converted to half-size characters are specified in the character string or null is specified, the specified characters are reported without change.

MojiPostiion

<<Function>>   Reports the position of the specified character in the character string with the number of characters from the beginning.

<<Coding>>   MojiPosition(character string,byte position)

<<Argument>>

Character string = Specify the character string of which character position is to be known.

Byte position = Specify the byte position with 0-relative (with the number of bytes from the beginning).

<<Return Value>>   The character position of the specified character (0-relative) is reported.

If null is specified in the character string, -1 is reported.   If the byte position is longer than the character string length, null is reported.

Date

<<Function>>   Posts a date in a string.

<<Coding>>   Date()

<<Argument>>   None

<<Return Value>>   The date is returned in the "yyyy/mm/dd" format.   "yyyy" is a year. "mm" is a month (01 to 12).   "dd" is a day (01 to 31).

Day

<<Function>>   Posts a day in a value.

<<Coding>>   Day()

<<Argument>>   None

<<Return Value>>   The day is returned in a value in the range of 1   to 31.

DayOfWeek

<<Function>>   Posts a day of the week in a value.

<<Coding>>   DayOfWeek()

<<Argument>>   None

<<Return Value>>   A day of the week is returned in a value.   Days of the week are assigned the values shown below.

Value	::	Day of the week"
0	:	"Sun."
1	:	"Mon."
2	:	"Tue."
3	:	"Wed"
4	:	"Thu."
5	:	"Fri."
6	:	"Sat."

Month

<<Function>>   Posts a month in a value.

<<Coding>>   Month()

<<Argument>>   None

<<Return Value>>   A month is returned in a value in the range of 1   to 12.

Year

<<Function>>   Posts a year in a value.

<<Coding>>   Year()

<<Argument>>   None

<<Return Value>>   A year is returned in a value.

Time

<<Function>>   Posts a time in a character string.

<<Coding>>   Time()

<<Argument>>   None

<<Return Value>>   A time is returned in the "hh/mm/ss" format.   "hh" is hours (00 to 23). "mm" is minutes (00 to 59).   "ss" is seconds (00 to 59).

ToSeconds

<<Function>>   Converts a time expressed in the "hh:mm:ss" format into a time in seconds starting from "00:00:00."

<<Coding>>   ToSeconds(Time)

<<Argument>>

Specifies a time to be converted into seconds.   Available values are as follows:

Hours: 00 to 23

Minutes: 00 to 59

Seconds: 00 to 59

<<Return Value>>   A time in seconds is returned.

ToDays

<<Function>>   Converts a date expressed in the "yyyy:mm:dd" format into a time in serial days starting from "1900:01:01" (January 01, 1900).

<<Coding>>   ToDays(Date)

<<Argument>>   Specifies a time to be converted into serial days.   Available values are as follows:

Year: 1900 to 2200

Month: 01 to 12

Day: 01 to 31

<<Return Value>>   A time in serial days is posted.

ToDate

<<Function>>   Converts a date expressed in serial days starting from "1900:01:01" (January 01, 1900) into a date expressed in the "yyyy:mm:dd" format.

<<Coding>>   ToDate(SerialDays)

<<Argument>>   SerialDays = Specifies a time in serial days (1 to 109938) into a date.

<<Return Value>>   A date is posted.

ToTime

<<Function>>   Converts a time expressed in serial seconds starting from "00:00:00" into a 24hour-system time expressed in the "hh:mm:ss" format.

<<Coding>>   ToTime(SerialSeconds)

<<Argument>>   SerialSeconds = Specifies a time in serial seconds (0 to 86399) into a date.

<<Return Value>>   A time is posted.

FormatDate

<<Function>>   Converts the format of a date into a specified date format.

<<Coding>>   FormatDate(Format,Date)

<<Argument>>

Format = Specifies one of the date formats listed below. Uppercase   letters indicate that leading zeros can be left blank.

NN	:	Era name (2 full-size characters)
yy,yyyy,YY,YYYY	:	Year position (2 or 4 half-size characters)
mm,MM	:	Month position (2 half-size characters) dd,DD: Day position (2 half-size characters)
W	:	Day-of-Week position (1 full-size character)

Date = Specifies a date to be converted in the "yyyy:mm:dd" format. Available values are as follows:

yyyy	:	1900 to 2200
mm	:	01 to 12
dd	:	01 to 31

<<Return Value>>   An edited date is posted.

<<Example>>   A specified date of "1993/10/01" can be edited as shown below according to date formats for edition.

Date format	:	Result of conversion
"yyyy/mm/dd"	:	"1993/10/01"
"yyYearmmMonthddDay"	:	"93Year10Month01Day"
"MMDDYY(,v)"	:	"10 193((Fri.))"
",m,mYYYear"	:	"HEISEI 5Year"

FormatTime

<<Function>>   Converts the format of a time into a specified time format.

<<Coding>>   FormatTime(Format,Time)

<<Argument>>

Format = Specifies one of the time formats listed below. Uppercase   letters indicate that leading zeros can be left blank.

hh,HH	:	"Hours" position (in 24- or 12-hours system, 2 half-size characters)
tt,TT	:	"Hours" position (in 24- or 12-hours system, 2 half-size characters)
mm,MM	:	"Minutes" position (2 half-size characters)
ss,SS	:	"Seconds" position (2 half-size characters)
!!nn,NN,XX!!	:	Position of a character string representing morning or afternoon (nn: am/pm, NN : AM/PM, !!XX: XX/XX!!)

Time = Specifies a time (00:00:00 to 23:59:59) to be converted in the 24-hour system "hh:mm:ss" format.

<<Return Value>>   An edited time is posted.

<<Example>>   A specified time of "13:05:48" can be edited as shown below according to date formats for edition.

Time format	:	Result of conversion
"hh:mm:ss"	:	"13:05:48"
"HHhMMmSSs"	:	"13h 5m48s"
"mmhh"	:	"0513"
"!!"XXTTXX"!!"	:	"!!"XX 1XX"!!"
"NNHH:MM"	:	"PM 1: 5"
"HHhh"	:	"13hh"

Beep

<<Function>>   Causes the buzzer to beep.

<<Coding>>   Beep()

<<Argument>>   None

<<Return Value>>   None

Debug

<<Function>>   Causes the debugger to start.

<<Coding>>   Debug ()

<<Argument>>   None

<<Return Value>>   None

Cd

<<Function>>   Changes a working directory.

<<Coding>>   Cd(Path-name)

<<Argument>>   Path-name =   Specifies the path name of the working directory which you want to change.

<<Return Value>>   "True" is returned when the working directory is changed successfully.

ExitScript

<<Function>>   Exits the script.

<<Coding>>   ExitScript()

<<Argument>>   None

<<Return Value>>   None

CloseTask

<<Function>>   Deletes (or forcibly exits) a program.

<<Coding>>   CloseTask(ObjectID)

<<Argument>>   ObjectID = Specifies the object ID of a program which you want to delete.

<<Return Value>>   None

ExitWindows

<<Function>>   Forcibly exits a Windows system.

<<Coding>>   ExitWindows()

<<Argument>>   None

<<Return Value>>   A null is posted when there is an application which rejects the termination of the Windows system.

<<Note>>   This function does not return when the Windows system ends normally.

REfEnv

<<Function>>   Enables reference of environmental variables.

<<Coding>>   RefEnv(variable-name)

<<Argument>>   variable-name =   Specifies the name of a variable you want to read.

<<Return Value>>   The contents of the specified environmental variable is posted.   When the specified environmental variable is not found, a character string of length 0 is posted.

Wait

<<Function>>   Waits until the specified object ends.

<<Coding>>   Wait(ObjectID)

<<Argument>>   Wait(ObjectID) =   Specifies the object ID of a task which waits.

<<Return Value>>   The return value of the task whose termination is a waited is posted.

<<Win32>>   When the object ID of a Win16 application is specified, this function will not be returned even when the Win16 application ends.   Call Wait16 to wait for termination of a Win16 application.

Drive

<<Function>>   Changes a working drive.

<<Coding>>   Drive(Drive-name)

<<Argument>>   Drive-name =   Specifies the new drive name.

<<Return Value>>   "True" is posted when the drive is changed successfully.

WindowsInformation

<<Function>>   Posts Windows information.

<<Coding>>   WindowsInformation(Information-name)

<<Argument>>   Information-name =   Specifies information you want to know.

1: Quantity of available memory

2: System CPU

3: Operation mode of Windows

4: Path name of Windows directory

5: Current version of Windows

6: Font size

7: Screen information

<<Return Value>>   Windows information is posted.

Tasklist

<<Function>>   Reads a list of tasks.

<<Coding>>   TaskList()

<<Argument>>   None

<<Return Value>>   A list of the loaded tasks is posted in an array.

GetObjectID

<<Function>>   Gets an object ID having a specified Windows title.

<<Coding>>   GetObjetID(<title<,type>>)

<<Argument>>

title = Specifies the Windows title whose object ID you want to get.   When this argument is omitted, it is assumed that the title of an active Windows program is specified.

Type = Specifies a type of obtaining the specified object ID: by a path (PATH) or by a Windows title (CAUTION).   When this argument is omitted, it is assumed that "CAUTION" is specified.

<<Return Value>>   An obtained object ID is posted.

<<Win32>>   When "PATH" is specified as a "type" argument, the object ID cannot be obtained.

GetProfile

<<Function>>   Gets character strings described in WIN.INI.

<<Coding>>   GetProfile(Information-name)

<<Argument>>   Information-name = Specifies an information name to be loaded in the "Section, Keyword" format.

<<Return Value>>   When a keyword exists, its strings are posted.   If not, a character string of length 0 is posted.

<<Win32>>   WIN.INI information being mapped in the registry is loaded.

PostMessage

<<Function>>   Sends messages to a specified Window.

<<Coding>>   PostMessage(Object-ID,message-number<word-parameter><higher-word><lower-word>)

<<Argument>>

Object-ID = Specifies the object ID of a Window which you want to sent a message.

Message-number = Specifies the number of a message to be sent to the Windows.   The message number is the value (in decimal) of a WM_message defined in WINDOWS.H.

Word-parameter = Specifies a word parameter related to the message number.   When this parameter is omitted, it is assumed that 0 is specified.   higher-word = Specifies the higher word of a long parameter related to the message number.   When this parameter is omitted, it is assumed that 0 is specified.

lower-word = Specifies the lower word of a long parameter related to the message number. When this parameter is omitted, it is assumed that both the higher and lower words are specified for "higher-word."

<<Return Value>>   "true" is posted when a message is sent successfully, and "false" is posted when a message is not sent.

<<Note>>   Refer to the Programmer's Reference for details of word parameter, higher-word, and lower-word values.

VarType

<<Function>>   Posts the type of an operational result of a specified expression.

<<Coding>>   VarType(Expression)

<<Argument>>   Expression = Specifies an expression or value whose type you want to know.

<<Return Value>>   A type string corresponding to the value or operational result of the specified expression is posted.

Type string returned	Type
"STR"	String type
"NUM"	Number type
"NULL"	Null type
"OBJ"	Object ID type
"ARRAY"	Array type

IsString

<<Function>>   Judges whether the specified expression is of a string type.

<<Coding>>   IsString(expression)

<<Argument>>   expression = Specifies an expression or value to be judged.

<<Return Value>>   "true" is posted when the value or operational result of the specified expression is of a string type.   "false" is posted when it is not of the string type.

IsNum

<<Function>>   Judges whether the specified expression is a number type.

<<Coding>>   IsNum(expression)

<<Argument>>   expression = Specifies an expression or value to be judged

<<Return Value>>   "true" is posted when the value or operational result of the specified expression is a number type.   "false" is posted when it is not a number type.

IsNull

<<Function>>   Judges whether the specified expression is a null type.

<<Coding>>   IsNull(expression)

<<Argument>>   expression = Specifies an expression or value to be judged

<<Return Value>>   "true" is posted when the value or operational result of the specified expression is a null type.   "false" is posted when it is not a null type.

IsObj

<<Function>>   Judges whether the specified expression is an object ID type.

<<Coding>>   IsObj(expression)

<<Argument>>   expression = Specifies an expression or value to be judged

<<Return Value>>   "true" is posted when the value or operational result of the specified expression is an object ID type.   "false" is posted when it is not an object ID type.

IsArray

<<Function>>   Judges whether the specified expression is an array type.

<<Coding>>   IsArray(expression)

<<Argument>>   expression = Specifies an expression or value to be judged

<<Return Value>>   "true" is posted when the value or operational result of the specified expression is an array type.   "false" is posted when it is not of an array type.

DDEInitiate

<<Function>> Initiates DDE conversation with a specified server. When there are two or more servers, a dialog box appears for selection of a server.

<<Coding>> DDEInitiate(<application-name><,topic-name>)

<<Argument>>

Application-name = Specifies the name of a target application. When this argument is omitted, a dialog box appears for selection of an application.

Topic-name = Specifies the name of a topic. When this argument is omitted, a dialog box appears for selection of a topic.

<<Return Value>> When the specified server is initiated successfully, the object ID of the initiated server is posted.

Terminate

<<Function>>   Terminates DDE conversation which was initiated with the InitiateDDE function.

<<Coding>>   DDE-ID.Terminate()

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>   None

<<Return Value>>   None

Poke

<<Function>>   Sends data to the selected server using the DDE conversation which was initiated with the InitiateDDE function.   The data to be sent is internally converted into string type data and is sent as string data.

<<Coding>>   DDE-ID.Poke(data-name, send-data)

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>

data-name: Specifies the name of a data item.

Send-data: Specifies data to be sent.

<<Return Value>>   "True" is posted when data is sent successfully.

Request

<<Function>>   Requests the selected server to send data using the DDE conversation which was initiated with the InitiateDDE function.

<<Coding>>   DDE-ID.Request(item-name)

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>   item-name: Specifies the name of a data item supported by the server.

<<Return Value>>   The requested data is posted when the data request is processed successfully.

Execute

<<Function>>   Requests the selected server to execute a command using the DDE conversation which was initiated with the InitiateDDE function.

<<Coding>>   DDE-ID.Execute(command-name)

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>   command-name: Specifies the string of a command supported by the server.

<<Return Value>>   "true" is posted when the command is executed successfully.

Advise

<<Function>> Requests the selected server to initiate a hot link using the DDE conversation which was initiated with the InitiateDDE function. When receiving a hot link item, the server posts it to an application.

<<Coding>> DDE-ID.Advice(hot-link-item-name,event-reception-function-name)

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>

hot-link item name = Specifies the name of a hot link item.

event-reception-function-name = Specifies the name of an event reception function.

<<Return Value>> "true" is posted when the hot-link initiation request is executed successfully.

<<Note>> An error occurs when this function is executed by an application which does not support an event reception function.

You can refer to a hot link item with a DDE-ID item name by defining a post function. The post function is defined as shown below.

ret = funcname(item,value){...}

item	:	Hot-link item name
value	:	Hot-link item value
ret	:	No return value

When receiving hot-link data in the execution of a script, this function does not post to the application. The DDE conversation returns a "Busy" response.

This function is supported only when the server is a hot link. When the server is a warm link, use the DDE conversation in the post function to request the value.

Example)

```
funcname(item, value) {
                ret = o.request(item);
                ...
        }
```

Unadvise

<<Function>>   Terminates a hot link item initiated by the server using the DDE conversation which was initiated with the InitiateDDE function.

<<Coding>>   DDE-ID.Unadvice(hot-link-item-name)

Specify the return value of the DDEInitiate.DDEInitiate function as the DDE-ID.

<<Argument>>   hot-link item name =   Specifies the name of a hot link item.

<<Return Value>>   "true" is posted when the hot-link initiation item is terminated successfully.

<<Note>>   An error occurs when this function is executed by an application which does not support an event reception function.

DLLLoad

<<Function>> Loads a specified library.

<<Coding>> DLLLoad(Library)

<<Argument>> Library = Specifies the name of a DDL library to be loaded.

<<Return Value>> The object ID of a loaded library is posted.

DLLDeclare

<<Function>>   Defines functions (only functions that are pascal-declared) existing in a library loaded by the DLLLoad function as functions of a type that can be called by the script.

<<Coding>>   DLLDeclare(Object-ID,function-name,return-information,<,parameter-information>)

<<Argument>>

Object-ID = Specifies an object ID indicating a library loaded by the DLLLoad function.

Function-name = Specifies the name of a function to be declared.

Return-Information =   Specifies return information of the declared function.

Available information is as follows:

"NULL," "STRING," "CHAR," "SHORT," "LONG," "USHORT," "ULONG," "FLOAD," and "DOUBLE"

parameter-information =   Specifies parameter information of a declared function.   The information must be enclosed in double quotation marks.   When the declared function has two or more parameters,   They must be separated from each other by a comma and described in order.   The following parameter information can be specified:

CHAR<*:variable-name>,SHORT<*:variable-name>,LONG<*:variable-name>,USHORT<*:variable-name>,ULONG<*:variable-name>,FLOAT<*:variable-name>,DOUBLE<*:variable-name>,STRING<*:variable-name>,

A parameter with an asterisk is a parameter to be transferred to a pointer and the result is stored in the variable name following the asterisk.   The variable is stored in the object ID and can be referred to with "Object-ID.variable-name."

<<Return Value>>   "true" is posted when this function is processed normally.

<<Example>>   A function can be described as shown below for declaration:

```
 var object-ID;
 object-ID = DLLLoad("user-library");
 DLLDeclare(object-ID,"function","INT","STRING*5:variable-1,CHAR,INT");
```

<<Win32>>   Only stdcall-declared functions can be defined.   If the other functions are defined, their operation will not be assured.

DLLDelete

<<Function>>   Deletes a library which was loaded by the DLLLoad function.

<<Coding>>   DLLDelete(Object-ID)

<<Argument>>   Object-ID = Specifies an object ID indicating a library loaded by the DLLLoad function.

<<Return Value>>   "true" is posted when this function is processed normally.

DLLCalling

<<Function>>   Calls a function which was declared by the DLLDeclare function.

<<Coding>>   Object-ID.function-name(<parameter1><parameter2>...)

The "Object-ID" is the object ID indicating a library loaded by the DLLLoad function.

The "function-name" is defined by the DLLDeclare function.

<<Argument>>   parameter = Specifies according to the parameter information specified by the DLLDeclare function.   When this argument is omitted, it is assumed that all null-type parameters are specified.

<<Return Value>>   The return value of a called function is posted with a return information type specified by the DLLDeclare function.

<<Example>>   To call a function which is defined as "function-1" in the DLLLoad function, describe as follows:

```
var object-ID.result;object-ID = DLLLoad("user-library");
DLLDeclare(object-ID,"function-1","INT","STRING*5:variable-1,CHAR,INT"); result = object-ID.function-1("ABCDE","A",3);
```

Timer

<<Function>>   Halts processing until a specified time comes.

<<Coding>>   Timer(Time)

<<Argument>>   Time = Specifies a time (00:00:00 to 23:59:59 in the 24-hour system) at which the processing is restarted in the hh:mm:ss format.

<<Return Value>>   None

Sleep

<<Function>>   Halts processing for a specified number of seconds.

<<Coding>>   Sleep(number-of-seconds)

<<Argument>>   number-of-seconds = Specifies the number of seconds for which the processing is halted.

<<Return Value>>   None

DuplicateArray

<<Function>>   Copies a specified array.

<<Coding>>   DuplicateArray(array)

<<Argument>>   array = Specifies an array to be copied.

<<Return Value>>   The copied array is posted.

NumberArray

<<Function>>   Posts the number of items of a linear array or the number of items of the top row of a 2- or 3-dimensional array

<<Coding>>   NumberArray(array)

<<Argument>>   array = Specifies an array whose number of items you want to get.

<<Return Value>>   The number of items in the specified array is posted.

CreateArray

<<Function>>   Creates an array.

<<Coding>>   CreateArray(<initial-value>,number-of-dimensions-1<,number-of-dimensions-2>...)

<<Argument>>

initial-value =   Specifies an initial value to be set in the array.

number-of-dimensions-n =   Specifies the number of elements in the n-th dimension.   The "number-of-dimensions-1" value must be a positive integer.   When a value having a decimal point is specified, it is truncated to an integer.   When a null type value or 0 is specified, the succeeding specifications are all invalid.

<<Return Value>>   The specified array is posted.

<<Example>>   var array-1;

array-1 = CreateArray(null,2,3)

When this sample function is executed, the created array is identical to that defined by the following:

var array1[2][3]

Close

<<Function>>   Closes the Macro Manager in progress.   When this function is called in the script, the Terminate Function is called.

<<Coding>>   Close()

<<Argument>>   None

<<Return Value>>   None

DialogBoxOperatingFunction

- Mnemonic

  The dialog box operating function can have a mnemonic as a title. The mnemonic can be alphabetic characters preceded by an ampersand (&) or \036 or katakana characters preceded by \037. To display an ampersand, write two ampersands (&&). The following functions can have mnemonics:

  - AddTextBox.TextControl text
  - AddCheckBox.CheckBox text
  - AddPushButton.PushButton text
  - AddRadioButton.RadioButton text

  Below is shown a programming example to display a Mnemonic Set pushbutton.

  ```
  ID.AddPushButton("Mnemonic Set(&N)",0,0,85,10);
  RTN = ID.Show();
  ```

When a pushbutton is selected in the execution of the above program, the return value of the Show function is "<Mnemonic Set (N)."

- Variable name

  The dialog box operating function can specify variable names to be registered in a dialog box as arguments (string type). These variables are defined in the objects in the dialog box. To refer to them in the script, describe as shown below.

  - Dialog-box-object-ID.variable-name

CreateDialogBox

<<Function>>   Creates a dialog box object.

<<Coding>>   CreateDialogBox(title,X,Y,width,height)

<<Argument>>

title = Specifies a title to be displayed in the dialog box.

X =   Specifies the column position of the upper left corner of the dialog box.

Y =   Specifies the row position of the upper left corner of the dialog box.

width =   Specifies the width of a dialog box to be displayed.

height =   Specifies the height of a dialog box to be displayed.

<<Return Value>>   When the creation of a dialog box succeeds, the object ID of the created dialog box is posted.

<<Example>>var ID;

```
ID = CreateDialogBox("Sample",0,0,70,40);
ID.Show();
      . . .
```

AddText

<<Function>>   Registers a left-justification text control to the dialog box object which was created by the CreateDialogBox function.

<<Coding>>   Dig-ID.AddText(text,X,Y,width,height)

"Dig-ID" specifies the object ID of a dialog box to which a text control is registered.

<<Argument>>

text = Specifies a text to be displayed in the dialog box.   To display an ampersand, write two ampersands (&&).

X =   Specifies the column position of the upper left corner of the dialog box.

Y =   Specifies the row position of the upper left corner of the dialog box.

width =   Specifies the width of a dialog box to be displayed.

height =   Specifies the height of a dialog box to be displayed.

<<Return Value>>   None

<<Example>>var ID;

```
ID = CreateDialogBox("Sample",0,0,88,32);
ID.AddText("Text\nControl",4,8,80,16);
ID.Show();
        . . .
```

AddCheckBox

<<Function>>   Creates a check box and registers it to the dialog box object which was created by the CreateBox function.   A text specified by "Argument" is displayed to the right of the square mark !!"X"!!.   The result of a check box can be reconfirmed by referring to a variable after the dialog box is displayed.   When the value of a variable is "true," the checked status is displayed and when the value of a variable is "false," the unchecked status is displayed.

<<Coding>>   Dig-ID.AddCheckBox(title,X,Y,width,height,variable name)

"Dig-ID" specifies the object ID of a dialog box to which a check box is registered.

<<Argument>>

title = Specifies a title to be displayed to the right of a check box.

X =   Specifies the column position of the upper left corner of the check box.

Y =   Specifies the row position of the upper left corner of the check box.

width =   Specifies the width of a check box to be displayed.

height =   Specifies the height of a check box to be displayed.

variable-name = Specifies a variable name indicating the status of a check box with a string type.   The initial variable value is "false."   When the check box is clicked on, the variable value becomes "true."

<<Return Value>>   None

<<Example>>var ID;

```
ID = CreateDialogBox("Sample",0,0,88,64);
ID.AddCheckBox("check1(&1)",8,16,40,10,"C1");
ID.AddCheckBox("check2(&2)",8,26,40,10,"C2");
ID.AddCheckBox("check3(&3)",8,36,40,10,"C3");
ID.AddGroupBox("item-group1",4,4,48,48);
ID.Show();
      . . .
```

AddPushButton

<<Function>>　Creates a pushbutton which encloses a specified text in a box and registers it to the dialog box which was created by the CreateDialogBox function.　When any part in the pushbutton is clicked on, the dialog box disappears from the screen.

The result of the pushbutton is posted as a return value of the Show function.

<<Coding>>　Dig-ID.AddPushButton(title,X,Y,width,height,type)

"Dig-ID" specifies the object ID of a dialog box to which a pushbutton is registered.

<<Argument>>

text = Specifies a title to be displayed for the pushbutton.

X =　Specifies the column position of the upper left corner of a control to be displayed.

Y =　Specifies the row position of the upper left corner of a control to be displayed.

width =　Specifies the width of a control to be displayed.

height =　Specifies the height of a control to be displayed.

type = The type is a default pushbutton when "true" is specified.　When this argument is omitted or "false" is specified, the type is a normal pushbutton.

<<Return Value>>　None

<<Example>>var ID, return-value;

```
ID = CreateDialogBox("Sample",0,0,108,40);
ID.AddPushButton("button1(&A)",4,24,48,12,true);   <- 1
ID.AddPushButton("button2(&B)",56,24,48,12,false);   <- 2
return-value = ID.Show();
      . . .
```

AddRadioButton

<<Function>>   Creates a group box and a radio button which is filled in circle !!"X"!! when it is clicked on, and registers it to the dialog box which was created by the CreateDialogBox function.   The text specified by "argument" is displayed at the left of the radio button.

One or more radio buttons can be specified and they are displayed in the order they are specified.

The result of the radio buttons can be reconfirmed by referring to the variable after the dialog box is displayed.   When the value of the variable is "true," the result is the checked status.   When the value of the variable is "false," the result is the unchecked status.

<<Coding>>   Dig-ID.AddRadioButton(title,X,Y,width,height,variable-name,text1<,text2>...)

"Dig-ID" specifies the object ID of a dialog box to which a radio button is registered.

<<Argument>>

title = Specifies a title to be displayed in the group box.

X =   Specifies the column position of the upper left corner of a group box to be displayed.

Y =   Specifies the row position of the upper left corner of a group box to be displayed.

width =   Specifies the width of a group box to be displayed.

height =   Specifies the height of a group box to be displayed.

variable-name = Specifies the name of a variable indicating which radio button is selected with a character string.   The initial value is the text of a radio button which was specified first.

textn = Specifies a text to be displayed at the right of the radio button.

<<Return Value>>   None

<<Example>>var ID;

ID = CreateDialogBox("Sample",0,0,88,60);

ID.AddRadioButton("item-group",4,0,40,48,"R1","Item1","Item2","Item3");

ID.Show();

      . . .

AddEditText

<<Function>>   Creates an edit control which enables edition and entry of a text and registers it to a dialog box which was created by the CreateDialogBox function.

<<Coding>>   Dig-ID.AddEditText(X,Y,width,height,variable-name)

"Dig-ID" specifies the object ID of a dialog box to which an edit control is registered.

<<Argument>>

X =   Specifies the column position of the upper left corner of the edit control.

Y =   Specifies the row position of the upper left corner of the edit control.

width =   Specifies the width of the edit control to be displayed.

height =   Specifies the height of the edit control to be displayed.

variable-name = Specifies the name of a variable in which a text to be edited is stored.   The initial value is a character string ("") of length 0.

<<Return Value>>   None

<<Example>>var ID;

```
ID = CreateDialogBox("Sample",0,0,88,40);
ID.AddEditText(4,8,80,16,"E1");
ID.E1 = "initial-data";
ID.Show();
      . . .
```

AddGroupBox

<<Function>>   Creates a box which groups the other controls defined in the dialog box and registers it to the dialog box which was created by the CreateDialogBox function.

<<Coding>>   Dig-ID.AddGroupBox(title,X,Y,width,height)

"Dig-ID" specifies the object ID of a dialog box to which a group box is registered.

<<Argument>>

title = Specifies a title to be displayed in the group box.

X =   Specifies the column position of the upper left corner of the group box to be displayed.

Y =   Specifies the row position of the upper left corner of the group box to be displayed.

width =   Specifies the width of a group box to be displayed.

height =   Specifies the height of a group box to be displayed.

<<Return Value>>   None

<<Example>>var ID;

```
ID = CreateDialogBox("Sample",0,0,88,64);
ID.AddCheckBox("check1(&1)",4, 8,40,10,"C1");
ID.AddCheckBox("check2(&2)",4,18,40,10,"C2");
ID.AddCheckBox("check3(&3)",4,28,40,10,"C3");
ID.AddGroupBox("item-group1",0,0,40,40);
ID.Show();
      . . .
```

AddListBox

<<Function>>   Registers a list box to the dialog box object which was created by the CreateDialogBox function.

<<Coding>>   Dig-ID.AddListBox(X,Y,width,height,variable,array<,sort>

"Dig-ID" specifies the object ID of a dialog box to which a list box is registered.

<<Argument>>

X =   Specifies the column position of the upper left corner of a list box to be displayed.

Y =   Specifies the row position of the upper left corner of a list box to be displayed.

width =   Specifies the width of a list box to be displayed.

height =   Specifies the height of a list box to be displayed.

variable = Specifies the name of a variable which stores a selected character string.   The initial value is a character string specified for the 0th element of the array.

array = Specifies an array which stores a characterr string or value to be registered in the list box.

sort = Specifies whether a character string to be registered in the list box is sorted ("true") or not ("false").

<<Return Value>>   None

<<Example>>var ID;

```
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBox("Sample",0,0,88,50);
ID.AddListBox(6,8,30,24,"Result",string,true);
ID.Show();
      . . .
```

AddComboBox

<<Function>>   Registers a combination box in the object of a dialog box which was created by the CreateDialogBox function.

<<Coding>>   Dig-ID.ComboBox(X,Y,width,height,variable,array<,initial>)

"Dig-ID" specifies the object ID of a dialog box to which a combination box is registered.

<<Argument>>

X =   Specifies the column position of the upper left corner of a combination box to be displayed.

Y =   Specifies the row position of the upper left corner of a combination box to be displayed.

width =   Specifies the width of a combination box to be displayed.

height =   Specifies the height of a combination box to be displayed.

variable = Specifies the name of a variable which stores a selected string.   The initial value is a string specified in the first element of the array.

array = Specifies an array storing a string value to be registered in the combination box.

initial = Specifies an initial value to be displayed in the combination box.   A value to be specified as an initial value must already be specified in the array.

<<Return Value>>   None

<<Example>>

```
var ID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBox("Sample",0,0,88,64);
ID.AddComboBox(6,8,64,50,"Result",string,"string1");
ID.Show();
      . . .
```

Show

<<Function>>   Dislays a dialog box.   This function is released only when a pushbutton is pressed or "Close" is selected on the system menu.

<<Coding>>   Dig-ID.Show()

"Dig-ID" specifies the object ID of a dialog box to be displayed.

<<Argument>>   None

<<Return Value>>   A text (except for characters and parentheses specified for mnemonics) of the selected pushbutton is posted.   When "Close" on the system menu is selected, a string of length 0 is posted.

Delete

<<Function>>   Deletes an object of the dialog box.   In this case, both controls and variables defined in the object are also deleted.

<<Coding>>   Dig-ID.Delete()

"Dig-ID" specifies the object ID of a dialog box to be deleted.

<<Argument>>   None

<<Return Value>>   None

OpenFile

<<Function>>   Opens a specified file in a specified mode.

<<Coding>>   OpenFile(file-name,open-mode<,delimiter>)

<<Argument>>

file-name = Specifies the name of a file on a disk.

open-mode = Specifies a mode in which the file is opened.   Available file open modes are as follows:

READ: Read mode (An error occurs when the specified file is not found.)

WRITE: Write mode (When the specified file is not found, a new file is created.   When the specified file already exists on disk, the new file is written over the old file.   The contents of the old file are lost.)

APPEND = Append mode (When the specified file is not found, a new file is created.   When the specified file already exists on disk, new data is appended to the end of the existing file.

Delimiter = Specifies a character string to delimit data.   This argument is used to read data of the array type or to write two or more data items.

<<Return Value>>   When the specified file is opened successfully, the object ID indicating the file is posted.

Close

<<Function>>   Closes a file specified by "object-ID."

<<Coding>>   ObjectID.Close()

"ObjectID" specifies an object ID posted by the FileOpen function.

<<Argument>>   None

<<Return Value>>   "true" is posted when the specified file is closed successfully.

Read

<<Function>>   Read data from a file specified by "object-ID."   When data to be read is of the   string type, one line (before a NewLine character appears) is read.   The NewLine character is not included in a character string which is posted as a return value.

When data of the array type is read, data is delimited by a delimiter specified by the FileOpen function and stored in an array.   In this case, the delimiter characters are not stored in the array.

<<Coding>>   ObjectID.Read(<,mode>)

"ObjectID" specifies an object ID posted by the FileOpen function.

<<Argument>>   mode = Specifies a type of data to be read.

"STRING":   Reads data of the string type.

"ARRAY":   Reads data of the array type.

When this argument is omitted, it is assumed that "STRING" is specified.

<<Return Value>>   When reading succeeds, a character string or array read from the file is posted.

Write

<<Function>>   Writes data in a file specified by "object-ID."   NewLine characters are written after data is written in the file.

<<Coding>>   ObjectID.Write(write-data<,...)

"ObjectID" specifies an object ID posted by the FileOpen function.

<<Argument>>   write-data = Specifies a character string to be written in a specified file. When two or more data items are specified, they are written with items delimited by a delimiter which was specified by the FileOpen function.

<<Return Value>>   When writing succeeds, "true" is posted.

Eof

<<Function>>   Checks whether the end of a file specified by "object-ID is reached.

<<Coding>>   ObjectID.Eof()

"ObjectID" specifies an object ID posted by the FileOpen function.

<<Argument>>   None

<<Return Value>>   When the end of the file has already been reached, "true" is posted.
When the end of the file has not been reached, "false" is posted.

FileList

<<Function>>   Reads a list of file names.

<<Coding>>   FileList(<file>)

<<Argument>>   file = Specifies file names (or a wild card) or a path name to be read. When this argument is omitted, all files in the current directory are read.

<<Return Value>>   A list of the read file names (array type) is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

CopyFile

<<Function>>   Copies a specified file.

<<Coding>>   CopyFile(file1,file2)

<<Argument>>

file1 = Specifies a file name (or a wild card) or path name to be copied.

file2 = Specifies a file name or directory name to which a specified file is copied.

<<Return Value>>   When file copying succeeds, "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file1" and "file2."

DeleteFile

<<Function>>   Deletes a specified file.

<<Coding>>   DeleteFile(file)

<<Argument>>   file = Specifies a file name (or a wild card) or path name to be deleted.

<<Return Value>>   When a specified file is deleted successfully, "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

RenameFile

<<Function>>   Renames a specified file.

<<Coding>>   RenameFile(file1,file2)

<<Argument>>

file1 = Specifies a file name (or a wild card) to be renamed.

file2 = Specifies a new file name (or a wild card).

<<Return Value>>   When a specified file is renamed successfully, "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file1" and "file2."

MakeDirectory

<<Function>>   Creates a subdirectory.

<<Coding>>   MakeDirectory(directory)

<<Argument>>   directory = Specifies the name of a directory (or a path name) to be created.

<<Return Value>>   When a specified directory is created successfully, "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "directory."

RemoveDirectory

<<Function>>   Deletes a specified subdirectory.

<<Coding>>   RemoveDirectory(directory)

<<Argument>>   directory = Specifies the name of a directory (or a path name) to be deleted.

<<Return Value>>   When a specified directory is deleted successfully, "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "directory."

RemoveDirectoryAll

<<Function>>   Deletes a specified subdirectory, all files in the directory, and the subdirectory.

<<Coding>>   RemoveDirectoryAll(directory)

<<Argument>>   directory = Specifies the name of a directory (or a path name) to be deleted.

<<Return Value>>   When the specified directory deletion succeeds,   "true" is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "directory."

ExistFile

<<Function>>   Checks whether the specified file exists.

<<Coding>>   ExistsFile(File)

<<Argument>>   file = Specifies the name of a file (or wild card) or a path name whose existence you want to check.

<<Return Value>>   "true" is posted when the specified file exists or "false" is posted when the specified file does not exist.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

FileAttribute

<<Function>>   Posts the attribute of a specified file.

<<Coding>>   FileAttribute(File)

<<Argument>>   file = Specifies the name of a file (excluding a wild card) or a path name whose attribute you want to check.

<<Return Value>>   When this function ends successfully, the following value is posted according to the attribute of the file:

"NORMAL":   Normal file

"READ":   Read-only file "DIRECTORY":   Directory

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

GetDrive

<<Function>>   Gets a drive name from a specified file name.

<<Coding>>   GetDrive(File)

<<Argument>>   file = Specifies the name of a file (or a wild card) or a path name from which you want to get a drive name.

<<Return Value>>   When a drive name exists, the drive name is posted.   When a drive name is not found, a character string of length 0 is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

GetPath

<<Function>>   Gets a path name from a specified file name.

<<Coding>>   GetPath(File)

<<Argument>>   file = Specifies the name of a file (or a wild card) or a path name from which you want to get a path name.

<<Return Value>>   When a path name exists, the path name is posted.   When a path name is not found, a character string of length 0 is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

GetFileName

<<Function>>   Gets a file name from a specified file name.

<<Coding>>   GetFileName(File)

<<Argument>>   file = Specifies the name of a file (or a wild card) or a path name from which you want to get a file name.

<<Return Value>>   When a file name exists, the file name is posted.   When a file name is not found, a character string of length 0 is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

GetName

<<Function>>   Gets a name from a specified file name.

<<Coding>>   GetName(File)

<<Argument>>   file = Specifies the name of a file (or a wild card) or a path name from which you want to get a name.

<<Return Value>>   When a name exists, the name is posted.   When a name is not found, a character string of length 0 is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

GetExtention

<<Function>>   Gets an extension from a specified file name.

<<Coding>>   GetExtension(File)

<<Argument>>   file = Specifies the name of a file (or a wild card) or a path name whose extension you want to get.

<<Return Value>>   When an extension exists, the extension is posted.   When an extension is not found, a character string of length 0 is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."   When a character string such as "abc.def.ghi" is specified for "file," "ghi" is posted as the extension.

SuitableName

<<Function>>   Posts a name which is not found in the working directory.

<<Coding>>   SuitableName(<file>)

<<Argument>>   file = Specifies a path name or name of a file (or a wild card).   When a file name is specified, a name except the wild card part is generated.

<<Return Value>>   A generated name is posted.

<<Win32>>   A character string of up to 259 bytes long can be specified for "file."

SetAttribute

<<Function>>   Sets the attribute of a specified file.

<<Coding>>   SetAttribute(file,file-attribute)

<<Argument>>   file = Specifies a path name or file name (or a wild card).

file-attribute = Specifies the attribute of a specified file using a flag shown below.   More than one attribute can be set at a time by combining the flags.

Flag	Attribute
0	Normal file
1	Read-only file
2	Hidden file
4	System file

<<Return Value>>   "true" is posted when processing ends successfully.

<<Note>>   When an attribute is changed, all old attributes are lost.   To keep the old attributes, specify their flags together with the new flag(s).

RestoreWindow

<<Function>>   Redisplays a window for the other minimized or maximized tool.

<<Coding>>   RestoreWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be redisplayed.

<<Return Value>>   None

MoveWindow

<<Function>>   Moves a window for the other tool.

<<Coding>>   MoveWindow(object-ID<,<column><row>>)

<<Argument>>

object-ID = Specifies the object ID of a task having a window to be moved.

column = Specifies the column position (-32767 to 3767) of the upper left corner of a window to be moved.

row = Specifies the row position (-32767 to 3767) of the upper left corner of a window to be moved.

When both "column" and "row" attributes are omitted, or null types are specified, the window can be moved using the cursor.

<<Return Value>>   None

SizeWindow

<<Function>>   Changes the size of a window for the other tool.

<<Coding>>   SizeWindow(object-ID<,<width><height>)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be re-sized.

width = Specifies the new width (-32767 to 3767) of a window to be re-sized.

height = Specifies the height (-32767 to 3767) of a window to be re-sized.

When both "width" and "height" attributes are omitted, or null types are specified, the window can be sized using the cursor.

<<Return Value>>   None

MaximizeWindow

<<Function>>   Maximizes a window for the other tool.

<<Coding>>   MaximizeWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be maximized.

<<Return Value>>   None

MinimizeWindow

<<Function>>   Minimizes a window for the other tool.

<<Coding>>   MinimizeWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be minimized.

<<Return Value>>   None

CloseWindow

<<Function>>   Closes a window for the other tool.

<<Coding>>   CloseWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be closed.

<<Return Value>>   None

ActivateWindow

<<Function>>   Activates a window for another tool.

<<Coding>>   ActivateWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window to be closed.

<<Return Value>>   None

GetWindowPos

<<Function>>   Gets the position of a window for the other tool.   The window position is represented by on-screen coordinates relative to the upper left corner (origin) of the screen.

<<Coding>>   GetWindowPos(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window whose position will be got.

<<Return Value>>   The position of the window which is obtained is posted in the following format:

X-coordinate: 0th element of array

Y-coordinate: 1st element of array

GetWindowSize

<<Function>>   Gets the size (width and height) of a window for another tool.

<<Coding>>   GetWindowSize(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window whose size will be obtained.

<<Return Value>>   The size of the window which is obtained is posted in the following format:

Width: 0th element of array

Height: 1st element of array

BottomWindow

<<Function>>   Place the window for another tool on the bottom of a screen of windows.

<<Coding>>   BottomWindow(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a task having a window which is placed on the bottom.

<<Return Value>>   None

ClipboardPaste

<<Function>>   Reads strings (a maximum of 64000 bytes) from the clipboard.

<<Coding>>   ClipboardPaste()

<<Argument>>   None

<<Return Value>>   The strings read from the clipboard are posted.

ClipboardCopy

<<Function>>   Copies the specified strings onto the clipboard.

<<Coding>>   ClipboardCopy(string)

<<Argument>>   string = Specifies a one or more character strings to be copied onto the clipboard.

<<Return Value>>   "true" is posted when the copying succeeds.

BitAnd

<<Function>>   Performs an AND on two specified values.

<<Coding>>   BitAnd(value1, value2)

<<Argument>>   value1 = Specifies a value to be ANDed.

value2 = Specifies a value for which an AND operation is to be performed.

<<Return Value>>   The result of AND of the specified values is posted.

<<Example>>

```
BitAnd(0, 0);   ->       0
BitAnd(1, 0);   ->       0
BitAnd(1, 1);   ->       1
```

<<Note>>   When values having a decimal point are specified as "value1" and "value2", they are truncated and then an OR operation is performed.

BitNegate

<<Function>>   Reverses the bits of a specified value.   The result is the value of the opposite sign minus 1.

<<Coding>>   BitNegate(value)

<<Argument>>   value = Specifies a value whose bits are reversed.

<<Return Value>>   The result of negation is posted.

<<Example>>

```
BitNegate(1);  ->       -2
BitNegate(-1); ->        0
```

<<Note>>   When a value having a decimal point is specified as "value," it is truncated and negated.

BitOr

<<Function>>   Performs an OR operation on two specified values.

<<Coding>>   BitOr(value1, value2)

<<Argument>>

value1 = Specifies a value for which an OR operation is to be performed.

value2 = Specifies a value for which an OR operation is to be performed.

<<Return Value>>   The result of OR of the specified values is posted.

<<Example>>

```
BitOr(0, 0);     ->       0
BitOr(1, 0);     ->       1
BitOr(1, 1);     ->       1
```

<<Note>>   When values having a decimal point are specified as "value1" and "value2," they are truncated and an OR operation is performed.

BitShiftLeft

<<Function>>   Arithmetically shifts left a specified value by a specified number of bits.

<<Coding>>   BitShiftLeft(value,number-of-shift-bits)

<<Argument>>

value = Specifies a value to be shifted left arithmetically.

number-of-shift-bits = Specifies the number of bits by which the specified value is shifted left.

<<Return Value>>   The result of shifting is posted.

<<Example>>

```
BitShiftLeft(1, 1);       ->     2
BitShiftLeft(1, 2);       ->     4
BitShiftLeft(-1, 2);      ->    -4
```

<<Note>>   When a value having a decimal point is specified as "value," they are truncated and shifted.

A value over 32 specified as "number-of-shift-bits" is assumed to be 32.

BitShiftRight

<<Function>>   Arithmetically shifts right a specified value by a specified number of bits.

<<Coding>>   BitShiftRight(value,number-of-shift-bits)

<<Argument>>   value = Specifies a value to be shifted right arithmetically.

number-of-shift-bits = Specifies the number of bits by which the specified value is shifted right.

<<Return Value>>   The result of shifting is posted.

<<Example>>

```
BitShiftRight(2, 1);     ->        1
BitShiftRight(4, 2);     ->        1
BitShiftRight(-4, 2);    ->       -1
```

<<Note>>   When a value having a decimal point is specified as "value," they are truncated and shifted.

A value over 32 specified as "number-of-shift-bits" is assumed to be 32.

BitXor

<<Function>>   Performs an exclusive OR operation on the two specified values.

<<Coding>>   BitXor(value1, value2)

<<Argument>>   value1 = Specifies a value for which an exclusive OR operation is to be performed.

value2 = Specifies a value for which an exclusive OR operation is to be performed.

<<Return Value>>   The result of the XOR operation of the specified values is posted.

<<Example>>

```
BitXor(0, 0);   ->      0
BitXor(1, 0);   ->      1
BitXor(1, 1);   ->      0
```

<<Note>>   When values having a decimal point are specified as "value1" and "value2," they are truncated and an XOR operation is performed.

Car

<<Function>>   Gets the top Item of a specified List.

<<Coding>>   Car(List<,delimiter-string>)

<<Argument>>

List = Specifies a list whose top item you want to obtain.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   The fetched top item is posted.

<<Note>>   When a list is null, a null is posted.

Cdr

<<Function>>   Deletes the top Item of a specified List.

<<Coding>>   Cdr(List<,delimiter-string>)

<<Argument>>

List = Specifies a list whose top item you want to delete.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   The list without the top item is posted.

<<Note>>   When a list is null, a null is posted.

Element

<<Function>>   Gets an Item at a specified position of a specified List.

<<Coding>>   Element(list,fetch-position<,delimiter-string>)

<<Argument>>

list = Specifies a list from which you want to get an item.

fetch-position = Specifies a position of an item to be fetched in the list relative to the 0th element of the list.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   The item fetched from a specified position is posted.

<<Note>>   When a list is null, a null is posted.

ElementPos

<<Function>>   Gets the position (or element number) of a specified Item in a specified List.   When the list has two or more identical items, the position of the item first obtained is posted.

<<Coding>>   ElementPos(list,search-item<,delimiter-string>)

<<Argument>>

list = Specifies a list in which you want to get the position of an item.

search-item = Specifies an item whose position in the list you want to know.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   The position of the specified item (relative to the 0th element) in the list is posted.

<<Note>>   When a list is null, "-1" is posted.

GetElementsAll

<<Function>>   Gets all Items in a specified List.

<<Coding>>   GetElementsAll(list<,delimiter-string>)

<<Argument>>

list = Specifies a list, all of whose items you want to get.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   An array storing all items fetched from the list is posted.

<<Note>>   When "..." is specified for "list," the array has four items (elements), each of which stores a null.

List

<<Function>>   Concatenates strings stored in an array and creates a List.   In concatenation, strings are delimited with delimiters.

<<Coding>>   List(string<,delimiter-string>)

<<Argument>>

string = Specifies strings in an array with which a list is created.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   A created list is posted.

NumElements

<<Function>>   Gets the number of Items(elements) in a specified List.

<<Coding>>   NumElements(list<,delimiter-string>)

<<Argument>>

list = Specifies a list the number of whose items you want to get.

delimiter-string = Specifies a string to delimit strings.   When this argument is omitted, it is assumed that a comma is specified as a delimiting string.

<<Return Value>>   The number of items in the specified list is posted.

<<Note>>   When a list is null, 0 is posted as the number of items.

Eval

<<Function>>   Executes a specified script statement.

<<Coding>>   Eval(script-statement)

<<Argument>>   script-statement = Specifies a script statement to be evaluated.

<<Return Value>>   The return value of the script statement which is evaluated last is posted.

<<Example>>

```
var      func[2], i;
var      x, y, ret;
func[0] = "Min"; func[1] = "Max";
                    . . .
ret = Eval(Cat(func[i], "(x, y) ;"));
```

<<Note>>   The script statement cannot contain an "Eval" specification.

When "reference of Local Variable" is described in the script statement, a local variable which is valid when the "Eval" function is called is referred to.   When "declaration of local variable" is described in the script statement, call of the "Eval" function is processed in the same manner as the block description.

Example)

```
var      a;
a = NumToString(123);                                    // a = "123";
a = Eval("var a; a = 456; NumToString(a);");             // a = "456";
```

This is equivalent to the following script description:

```
var      a;
a = NumToString(123);                                    // a = "123";
var      tmp;
{
         var      a;
         a = 456;
         tmp = NumToString(a);
}
a = tmp;                                           // a = "456";
```

Getparameter

<<Function>>   Gets the parameters specified when a script is executed.

<<Coding>>   Getparameter()

<<Argument>>   None

<<Return Value>>   Arrays storing the parameter are posted. The contents of arrays are as follows:

[0]:   Number of parameters

[1 to n]:   Contents of parameters (when the parameters exist)

<<Example>>

```
 test(p1, p2, p3)
{
        var Array;
        Array = Getparameter();
                . . .
}
```

KeyBoard [Win16]

<<Function>>   Simulates input from the keyboard.

<<Coding>>   KeyBoard(object-ID,event)

<<Argument>>   object-ID = Specifies an object ID of a task which generates key and string events.

event = Specifies strings corresponding to key and string events to be generated.   A string enclosed in angle brackets ("<" and ">") generates a key event and the other string generates a string event.   Key events for pressing two keys at a time are concatenated by "+>."   Key events are as follows:

<CAN>: Pressing the Cancel key

<BACK>: Pressing the Backspace key

<TAB>: Pressing the Tab key

<RETURN>: Pressing the Return key

<SHIFT>: Pressing the Shift key

<CTRL>: Pressing the Ctrl key

<ALT>: Pressing the Alt key

<CAPS>: Pressing the Caps key

<ESC>: Pressing the Esc key

<SPACE>: Pressing the Space key

<PRIOR>: Pressing the Prior key

<NEXT>: Pressing the Next key

<HOME>: Pressing the Home key

<LEFT>: Pressing the Left key

<UP>: Pressing the Up key

<RIGHT>: Pressing the Right key

<DOWN>: Pressing the Down key

<INS>: Pressing the Insert key

<DEL>: Pressing the Delete key

<F1> to <F20>: Pressing the Function-n key

<A> to <Z>: Pressing the alphabetic keys (A to Z)

<0> to <9>: Pressing the numeric keys (0 to 9)

<EXECUTE>: Pressing the Execute key

The other keys are specified with virtual key codes <0x??>.

<SHIFT+>: Pressing an other key while holding down the Shift key

<CTRL+>: Pressing an other key while holding down the Ctrl key

<ALT+>: Pressing an other key while holding down the Alt key

A specification of "<XXX+>" and "<YYY>" can be replaced by a specification of "<XXX+YYY>."

<<Return Value>>   None

<<Example>>

KeyBoard(object-ID, "ABCabc<ALT+F><N>");

KeyBoard(object-ID, "<CTRL+SHIFT+F12>");

<<Note>>   To generate "<" by a string event, specify "<<."

Menu [Win16]

<<Function>>   Simulates menu operations.

<<Coding>>   Menu(object-ID,menu-item1<,menu-item2,...>)

<<Argument>>   object-ID = Specifies the object ID of a task having a menu item to be generated.   When a null type is specified for "object ID," events are generated on the currently active window.

menu-item1<,menu-item2,...> = Specifies strings representing menu items to be generated. Specify menu items in the numerically descending order starting from the main menu.   A string representing a menu item cannot contain a short-cut key "(?)."   Menu numbers (0 -) can be used instead of strings (which is valid when bit map menus are specified).

<<Return Value>>   None

<<Example>>   Menu(object-ID,"Minimize")

Mouse [Win16]

<<Function>>   Simulates mouse operations.

<<Coding>>   Mouse(object-ID,event,X,Y,<X1,Y1>)

<<Argument>>

object-ID = Specifies the object ID of a task for generating mouse events.   When a null type is specified for "object ID," events are generated on the currently-active window.

event = Specifies string corresponding to a mouse event to be generated.   Available strings are as follows:

LCLICK: Clicking the left button

LDOUBLE: Double-clicking the left button

LDRAG: Dragging with the left button pressed

RCLICK: Clicking the right button

RDOUBLE: Double-clicking the right button

RDRAG: Dragging with the right button pressed

X = Specifies an X-coordinate of a mouse position relative to the origin of the client area in the master window.

Y = Specifies a Y-coordinate of a mouse position relative to the origin of the client area in the master window.

X1 = Specifies an X-coordinate of a destination mouse position relative to the origin of the client area in the master window.   (Required when the mouse is dragged.)

Y1 = Specifies a Y-coordinate of a destination mouse position relative to the origin of the client area in the master window.   (Required when the mouse is dragged.)

<<Return Value>>   None

<<Example>>   Mouse(object-ID,"LDOUBLE",100,200)

Scrollbar [Win16]

<<Function>>   Simulates scroll-bar operations.

<<Coding>>   Scrollbar(object-ID,event<,X>)

<<Argument>>   object-ID = Specifies the object ID of a task for generating scroll-bar events.   When a null type is specified for "object ID," events are generated on the currently-active window.

event = Specifies string corresponding to a scroll-bar event to be generated.   Available strings are as follows:

LINEUP: Scrolling up by one line

LINEDOWN: Scrolling down by one line

LINELEFT: Scrolling left by one line

LINERIGHT: Scrolling right by one line

PAGEUP: Scrolling up by one page

PAGEDOWN: Scrolling down by one page

PAGELEFT: Scrolling left by one page

PAGERIGHT: Scrolling right by one page

XPOINT: Horizontal position (specified with a value)

YPOINT: Vertical position (specified with a value)

X = Specifies a value representing a scroll position.   (Specified only when "XPOINT" or "YPOINT" is specified as "event.")   The value can be a value starting from 0.   The maximum and the interval of this argument are dependent upon applications.

<<Return Value>>   None

<<Example>>   Scrollbar(object-ID,"LINEUP",100,200)

CreateDialogBoxEx

<<Function>>   Creates an extended dialog box object.

<<Coding>>   CreateDialogBoxEx(title,X,Y,width,height)

<<Argument>>   title = Specifies a title to be displayed in the extended dialog box.

X =   Specifies the column position of the upper left corner of the extended dialog box.

Y =   Specifies the row position of the upper left corner of the extended dialog box.

width =   Specifies the width of the extended dialog box to be displayed.

height =   Specifies the height of the extended dialog box to be displayed.

<<Return Value>>   When the creation of an extended dialog box succeeds, the object ID of the created extended dialog box is posted.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 40, 40);
```

AddTextEx

<<Function>>   Registers a text control to the extended dialog box object which was created by the CreateDialogBoxEx function.

<<Coding>>   Dig-ID.AddTextEx(text,X,Y,width,height[,style])

"Dig-ID" specifies the object ID of an extended dialog box to which a text control is registered.

<<Argument>>   text = Specifies a text to be displayed in the text control.

X =   Specifies the column position of the upper left corner of the text control to be displayed.

Y =   Specifies the row position of the upper left corner of the text control to be displayed.

width =   Specifies the width of a text control to be displayed.

height =   Specifies the height of a text control to be displayed.

style = Specifies the style of the text control in combination of the following values.   When this argument is omitted, it is assumed that a value of 0 is specified.

0: Left-adjusting a text

1: Center-adjusting a text

2: Right-adjusting a text

4: Not using an ampersand mark (&) as a prefix character

<<Return Value>>   None

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 40, 40);
ID.AddTextEx("Sample", 0, 0, 83, 32, 1);
```

AddCheckBoxEx

<<Function>>   Creates a check box and registers it to the extended dialog box object which was created by the CreateDialogBox function.   A text specified by "Argument" is displayed to the right of the square mark !!"XX"!!.   The result of a check box can be reconfirmed by referring to a variable after the extended dialog box is displayed.

When the value of a variable is "true," the checked status is displayed and when the value of a variable is "false," the unchecked status is displayed.

<<Coding>>   DigEx-ID.AddCheckBox(text,X,Y,width,height,variable-name[,style])

"DigEx-ID" specifies the object ID of an extended dialog box to which a check box is registered.<<Argument>>   title = Specifies a title to be displayed to the right of a check box.

X =   Specifies the column position of the upper left corner of the check box to be displayed.

Y =   Specifies the row position of the upper left corner of the check box to be displayed.

width =   Specifies the width of a check box to be displayed.

height =   Specifies the height of a check box to be displayed.

variable-name = Specifies a variable name indicating the status of a check box with a string type.   The following value is stored in the variable according to the status of the check box.

0: Not checked (Initial value)

1: Checked

2: Gray (light color) displayed

style = Specifies the style of the check box in combination of the following values.   When this argument is omitted, it is assumed that a value of 0 is specified.

0: Creating a 2-selection check box

1: Creating a 3-selection check box

2: Display a test to the left

<<Return Value>>   When processing ends successfully, the control ID is posted.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddCheckBoxEx("check1(&1)", 8, 16, 40, 10, "C1", 0);
ID.AddCheckBoxEx("check2(&2)", 8, 26, 40, 10, "C2", 0);
ID.AddCheckBoxEx("check3(&3)", 8, 36, 40, 10, "C3", 0);
ID.AddGroupBox("item-group1", 4, 4, 48, 48);
ID.ShowEx();
if (ID.C1 == 1) {
}
```

AddPushButtonEx

<<Function>>   Creates a push button which encloses a specified text in a box and registers it to the extended dialog box which was created by the CreateDialogBoxEx function.   Even when the push button is clicked, the dialog box will not disappear.

<<Coding>>   DigEx-ID.AddPushButtonEx(text,X,Y,width,height[,type])

DigEx-ID.ExtendedPushButton(text,X,Y,width,height[,type])

"DigEx-ID" specifies the object ID of a dialog box to which a push button is registered.

<<Argument>>   text = Specifies a text to be displayed in the push button.

X =   Specifies the column position of the upper left corner of a push button to be displayed.

Y =   Specifies the row position of the upper left corner of a push button to be displayed.

width =   Specifies the width of a push button to be displayed.

height =   Specifies the height of a push button to be displayed.

type = The type is a default push button when "true" is specified.   When this argument is omitted or "false" is specified, the type is a normal push button.

<<Return Value>>   When processing ends successfully, the control ID is posted.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddPushButtonEx("button1(&A)",   4, 24, 48, 12, true);
ID.AddPushButtonEx("button2(&B)", 56, 24, 48, 12, false);
```

AddRadioButtonEx

<<Function>>   Creates a group box and a radio button which is filled in circle !!"X"!! when it is clicked, and registers it to the extended dialog box which was created by the CreateDialogBoxEx function.   The text specified by "argument" is displayed to the right of the radio button.

One or more radio buttons can be specified and they are displayed downward in the order they are specified.

The result of the radio buttons can be reconfirmed by referring to the variable after the extended dialog box is displayed.   When the value of the variable is "true," the result is the checked status.   When the value of the variable is "false," the result is the unchecked status.

<<Coding>>   DigEx-ID.AddRadioButtonEx(text,X,Y,width,height,variable-name,text1[,text2,text3,..])

"DigEx-ID" specifies the object ID of an extended dialog box to which a radio button is registered.

<<Argument>>   text = Specifies a text to be displayed in the group box.

X =   Specifies the column position of the upper left corner of a group box to be displayed.

Y =   Specifies the row position of the upper left corner of a group box to be displayed.

width =   Specifies the width of a group box to be displayed.

height =   Specifies the height of a group box to be displayed.

variable-name = Specifies the name of a variable indicating which radio button is selected with a character string.   The initial value is the text of a radio button which was specified first.

textn = Specifies a text to be displayed to the right of the radio button.

<<Return Value>>   None

<<Note>>   At least two radio buttons must be required for selection.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddRadioButtonEx("item-group",   4, 0, 40, 48,
"R1", "Item1", "Item2", "Item3");
ID.ShowEx();
if (ID.R1 == "Item3") {         // When item-3 is selected
          //Processing of item-3
}
```

AddEditTextEx

<<Function>> Creates an edit control which enables edition and entry of a text and registers it to an extended dialog box which was created by the CreateDialogBoxEx function. The result of the edit control can be reconfirmed by referring to the variable after the extended dialog box is displayed. The entered string is stored in the variable.

<<Coding>> DigEx-ID.AddEditTextEx(X,Y,width,height,variable-name[.style])

"DigEx-ID" specifies the object ID of an extended dialog box to which an edit control is registered.

<<Argument>> X = Specifies the column position of the upper left corner of the edit control.

Y = Specifies the row position of the upper left corner of the edit control.

width = Specifies the width of the edit control to be displayed.

height = Specifies the height of the edit control to be displayed.

variable-name = Specifies the name of a variable in which a text to be edited is stored. The initial value is a character string ("") of length 0.

style = Specifies a style of the edit control in the combination of values shown below. When this argument is omitted, it is assumed that a combination of 0 and 256 is specified.

0: Text display position (left-justified)

1: Inputting two or more lines

2: Converting all entered characters into uppercase characters

4: Converting all entered characters into lowercase characters

8: Displaying all entered characters onscreen

16: Scrolling up by one page

32: Scrolling left/right by 10 characters (when data is entered on the end of line) or scrolling to the leftmost end of the line (when the Enter key is pressed)

64: Inhibiting input and edition of text

128: Making the Enter key work as a Carriage Return

256: Not displaying the outer frame

<<Return Value>> When processing ends successfully, the control ID is posted.

<<Example>>

```
var ID, input-data;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddEditTextEx(4, 8, 80, 16, "E1");
ID.E1 = "initial-data";
ID.ShowEx();
input-data = ID.E1;
```

AddGroupBoxEx

<<Function>>   Creates a box which groups the other controls defined on the extended dialog box and registers it to the extended dialog box which was created by the CreateDialogBoxEx function.

<<Coding>>   DigEx-ID.AddGroupBoxEx(text,X,Y,width,height)

"DigEx-ID" specifies the object ID of an extended dialog box to which a group box is registered.

<<Argument>>   text = Specifies a text to be displayed in the group box.

X =   Specifies the column position of the upper left corner of the group box to be displayed.

Y =   Specifies the row position of the upper left corner of the group box to be displayed.

width =   Specifies the width of a group box to be displayed.

height =   Specifies the height of a group box to be displayed.

<<Return Value>>   None

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddCheckBoxEx("check1(&1)", 4,   8, 40, 10, "C1");
ID.AddCheckBoxEx("check2(&2)", 4, 18, 40, 10, "C2");
ID.AddCheckBoxEx("check3(&3)", 4, 28, 40, 10, "C3");
ID.AddGroupBoxEx("item-group1", 0, 0, 40, 40);
```

ShowEx

<<Function>>   Displays an extended dialog box.   This function returnsimmediately after the screen is displayed, unlike the DialogBox Function.

<<Coding>>   DigEx-ID.ShowEx()

"DigEx-ID" specifies the object ID of an extended dialog box to be displayed.

<<Argument>>   None

<<Return Value>>   None

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddEditTextEx(4, 8, 80, 16, "E1");
ID.ShowEx();
```

DeleteEx

<<Function>>   Deletes an object of the extended dialog box.   In this case, both the controls and variables defined in the object are also deleted.

<<Coding>>   DigEx-ID.DeleteEx()

"DigEx-ID" specifies the object ID of the extended dialog box to be deleted.

<<Return Value>>   None

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
ID.AddEditTextEx(4, 8, 80, 16, "E1");
ID.ShowEx();
ID.DeleteEx();
```

AddListBoxEx

<<Function>>   Registers a list box to the extended dialog box object which was created by the CreateDialogBoxEx function.   The horizontal scroll bar is automatically displayed or not displayed according to the status of the item in the list box.   While directories are being displayed, you can move between them by double-clicking over the item.

<<Coding>>   DigEx-ID.AddListBoxEx(X,Y,width,height,variable-name,list[,style])

"DigEx-ID" specifies the object ID of an extended dialog box to which a list box is registered.

<<Argument>>   X =   Specifies the column position of the upper left corner of a list box to be displayed.

Y =   Specifies the row position of the upper left corner of a list box to be displayed.

width =   Specifies the width of a list box to be displayed.

height =   Specifies the height of a list box to be displayed.

variable-name = Specifies the name of a variable which stores a selected character string. The type of the variable is an array type.   The contents of arrays are as follows.   The value enclosed in parentheses is an initial value.

[0]: Number of selected items (0)

[1 - n]: Selected item (null type)

list = array type

When registering any item (string or value) in a list box, specify the array which stores the item.

string type

When registering a directory name in a list box, specify "directory."   When the other string is specified, an error occurs.

style = Specifies a style of a list box in the combination of values shown below.   When it is omitted, it is assumed that a value of 0 is specified.

0: Posting an entered message each time an item is clicked or double-clicked

1: Sorting in the alphabetic order

2: Enabling or disabling selection of an item by clicking

4: Enabling selection of two or more items by combining the Shift key and the mouse or a special key.

<<Return Value>>   When processing ends successfully, the control ID is posted.

<<Example>>

```
var ID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ID.AddListBoxEx(6, 8, 18, 48, "Result", string, 0);
ID.ShowEx();
if (ID.result[0] != 0) {          // The item in the list box is selected.
        if (ID.result[1] == "string1") {
                //The item of string-1 is selected.
                //Processing of string-1
        }
}
```

AddComboBoxEx

<<Function>>   Registers a combination box in the object of an extended dialog box which was created by the CreateDialogBoxEx function.

<<Coding>>   DigEx-ID.ComboBoxEx(X,Y,width,height,variable-name,combo[,initial-value[,style])>)

"DigEx-ID" specifies the object ID of an extended dialog box to which a combination box is registered.

<<Argument>>   X =   Specifies the column position of the upper left corner of a combination box to be displayed.

Y =   Specifies the row position of the upper left corner of a combination box to be displayed.

width =   Specifies the width of a combination box to be displayed.

height =   Specifies the height of a combination box to be displayed.

variable-name = Specifies the name of a variable which stores a selected item.   The initial value is a string of length 0 ("").

combo = array type

When registering an item (string or value) in a list box, specify the array which stores the item.

string type

When registering an item in a combination box, specify an array which stores the item.

string-type = Specify "true" when registering only a drive name in the combination box.   If another string is specified, an error occurs.

initial-value = Specifies a value which is initially displayed in the combination box.   When it is omitted, it is assumed that a string of length 0 ("") is specified.

style = Specifies a style of a combination box in the combination of values shown below.   If it is omitted, a value of 0 is specified.

0: Displaying the item selected from the list box in the edit control area

1: Sorting in alphabetic order

<<Return Value>>   None

<<Example>>

```
var ID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ID.AddComboBoxEx(6, 8, 18, 48, "result", string, "string1", 1);
ID.ShowEx();
if (ID.result== "string1") {
//The item of string-1 is selected.
//Processing of string-1
}
```

CloseDialogBox

<<Function>>   Closes an extended dialog box displayed on-screen.

<<Coding>>   DigEx-ID.CloseDialogBox()

"DigEx-ID" specifies the object ID of the extended dialog box to be deleted.

<<Argument>>   None

<<Return Value>>   A value of 1 is posted when the dialog box is closed successfully. A value of 0 is posted when the dialog box is already closed.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ID.ShowEx();
ID.CloseDialogBox();
```

ExistDialogBox

<<Function>>   Checks whether an extended dialog box is on-screen.

<<Coding>>   DigEx-ID.ExistDialogBox

"DigEx-ID" specifies the object ID of an extended dialog box whose display status is checked.

<<Argument>>   None

<<Return Value>>   A value of 1 is posted when the extended dialog box is displayed on-screen, and a value of 0 is posted when the extended dialog box is not on-screen.

<<Example>>

```
var ID;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ID.ShowEx();
if (ID.ExistDialogBox() == 1) {
//Processing when the dialog box is on-screen
}
```

AddListString

<<Function>>   Adds an item to a list box.

<<Coding>>   DigEx-ID.AddListString(control,item)

"DigEx-ID" specifies the object ID of an extended dialog box to which a list box is registered.

<<Argument>>   control = Specifies the control ID of a list box to which an item is added.

item = Specifies an item to be added to the list box.   When a null string is specified, the item is not added to the list box.

<<Return Value>>   A value of is posted when the item is added successfully, and a value of 0 is posted when the item is not added.

<<Note>>   An item to be added to a list box can be up to 64K bytes.   The position of an item to be added to the list box is as follows:

When sorted: The position of the added item is dependent upon its type.

When not sorted yet: The new item is added to the end of the existing item

<<Example>>

```
var ID, ListID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ListID = ID.AddListBoxEx(6, 8, 18, 48, "result", string, 0);
ID.AddListString(ListID, "string4");
ID.ShowEx();
```

DeleteListString

<<Function>>   Deletes an item from a list box.

<<Coding>>   DigEx-ID.DeleteListString(control,item)

"DigEx-ID" specifies the object ID of an extended dialog box to which a list box is registered.

<<Argument>>   control = Specifies the control ID of a list box from which an item is deleted.

item = Specifies an item to be deleted from the list box.   When a null string is specified, the item is not deleted from the list box.

<<Return Value>>   A value of 1 is posted when the item is deleted successfully, and a value of 0 is posted when the item is not deleted.

<<Example>>

```
var ID, ListID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ListID = ID.AddListBoxEx(6, 8, 18, 48, "result", string, 0);
ID.DeleteListString(ListID, "string1");
ID.ShowEx();
```

EnablePushButton

<<Function>>   Makes a pushbutton having a specified control ID selectable.   No operation is performed when the pushbutton is already selectable.

<<Coding>>   DigEx-ID.EnablePushbutton(control)

"DigEx-ID" specifies the object ID of an extended dialog box to which the pushbutton is registered.

<<Argument>>   control = Specifies the control ID of a pushbutton to be made selectable.

<<Return Value>>   A value of 1 is posted when the pushbutton is made selectable successfully, and a value of 0 is posted when the item is not made selectable.

<<Example>>

```
var ID, Push1, Push2;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
Push1 = ID.AddPushButtonEx("Pushbutton1(&A)",   4, 24, 48, 12, true);
Push2 = ID.AddPushButtonEx("Pushbutton2(&B)", 56, 24, 48, 12, false);
ID.ShowEx();
EnablePushButton(Push1);
EnablePushButton(Push2);
```

DisablePushButton

<<Function>>   Makes a pushbutton having a specified control ID unselectable.

No operation is performed if the pushbutton is already unselectable.

<<Coding>>   DigEx-ID.DisablePushbutton(control)

"DigEx-ID" specifies the object ID of an extended dialog box to which the pushbutton is registered.

<<Argument>>   control = Specifies the control ID of a pushbutton to be made unselectable.

<<Return Value>>   A value of 1 is posted when the pushbutton is made unselectable successfully, and a value of 0 is posted when the item is not made unselectable.

<<Example>>

```
var ID, Push1, Push2;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
Push1 = ID.AddPushButtonEx("Pushbutton1(&A)",   4, 24, 48, 12, true);
Push2 = ID.AddPushButtonEx("Pushbutton2(&B)", 56, 24, 48, 12, false);
ID.ShowEx();
EnablePushButton(Push1);
DisablePushButton(Push2);
```

GetPushButton

<<Function>> Posts which pushbutton is currently selected. The currently-selected pushbutton is the one which was selected last.

<<Coding>> DigEx-ID.GetPushbutton()

"DigEx-ID" specifies the object ID of an extended dialog box to which the pushbutton is registered.

<<Argument>> None

<<Return Value>> The control ID of the currently-selected pushbutton is posted. When no pushbutton is selected, a value of 0 is posted.

<<Example>>

```
var ID, Push1, Push2;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
Push1 = ID.AddPushButtonEx("Pushbutton1(&A)",   4, 24, 48, 12, true);
Push2 = ID.AddPushButtonEx("Pushbutton2(&B)", 56, 24, 48, 12, false);
ID.ShowEx();
EnablePushButton(Push1);
EnablePushButton(Push2);
if (ID.GetPushButton() == Push1) { //Button-1 is selected.
        //Button-1 processing
}
```

ResetPushButton

<<Function>>   Resets the currently selected pushbutton.

<<Coding>>   DigEx-ID.ResetPushbutton()

"DigEx-ID" specifies the object ID of an extended dialog box to which the pushbutton is registered.

<<Argument>>   None

<<Return Value>>   A value of 1 is posted when the currently-selected pushbutton is reset.

<<Example>>

```
var ID, Push1, Push2;
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 64);
Push1 = ID.AddPushButtonEx("Pushbutton1(&A)",   4, 24, 48, 12, true);
Push2 = ID.AddPushButtonEx("Pushbutton2(&B)", 56, 24, 48, 12, false);
ID.ShowEx();
EnablePushButton(Push1);
EnablePushButton(Push2);
if (ID.GetPushButton() == Push1) { //Button-1 is selected.
        //Button-1 processing
        ID.ResetPushButton() ;
}
```

ClearListSelect

<<Function>>   Deselect items which are selected in a list box.

<<Coding>>   DigEx-ID.ClearListSelect(control)

"DigEx-ID" specifies the object ID of an extended dialog box to which a list box is registered.

<<Argument>>   control = Specifies the control ID of a list box in which the selected items are deselected.

<<Return Value>>   A value of 1 is posted when the selected items are deselected successfully, and a value of 0 is posted when the selected items are not deselected.

<<Example>>

```
var ID, ListID;
var string[3];
string[0] = "string1";
string[1] = "string2";
string[2] = "string3";
ID = CreateDialogBoxEx("Sample", 0, 0, 88, 32);
ListID = ID.AddListBoxEx(6, 8, 18, 48, "result", string, 0);
ID.ShowEx();
if (ID.result[0] != 0) {          //The list box item is selected.
        if (ID.result[1] == "string1") {
        //The string-1 item is selected.
        //String-1 processing
        ID.ClearListSelect(ListID) ;
        }
}
```

Open16 [Win32]

<<Function>>   Opens a specified file.   When the specified file has an extension of .EXE, a Win16 application of the specified file name is executed.   When the specified file has another extension, the related Win16 application is started and a file of the specified file is opened.

The parameters specified in the parameter strings are transferred to the started Win16 application.

<<Coding>>   File-Name.Open(title<,parameter><,parameter>..)

"File-Name" specifies a file name of a file to be opened in a string form.

<<Argument>>   title =   Specifies a window title to be displayed in the master window of the Win16 application to be started.

parameter = Specifies a parameter to be transferred to the Win16 application specified by "File-Name."   The parameter can be up to 118 characters long.

<<Return Value>>   The object ID indicating the application is posted when the application is started successfully.

<<Note>>   When a master window having a window title is already opened, the object ID indicating the started Win16 application is not obtained in some cases.

To start a Win32 application, call the Open function.

To halt the termination of the Win16 application which was started by this function, call the Wait16 function.

<<Example>>

```
ver application;
application = "FILE01.EXE".Open("Win16application-window");      <- 1
application = ""DATA01.XJS".Open("Excel");      <- 2
```

When example 1 is executed, "FILE01.EXE" is started and the object ID indicating "FILE01.EXE" is set in the variable "application-ID."

When example 2 is executed, "MS-Excel" (if "DATA01.XJS" is related to "MS-Excel") is started, then "DATA01.XJS" is opened.   The object ID indicating "MS-Excel" is set in the variable "application-ID."

Wait16 [Win32]

<<Function>>   Halts the termination of the Win16 application of the specified object ID.

<<Coding>>   Wait16(object-ID)

<<Argument>>   object-ID = Specifies the object ID of a Win16 application to be made to wait.

<<Return Value>>   A value of 0 is posted when the processing is terminated normally.

<<Note>>   Unlike Wait this function cannot get an end code of the Win16 application.

Use this function to halt the termination of a Win16 application which was started by Open16."

About Script

To call "About Script," press the F1 key or select a "HELP" - "Use of HELP" command.

"Tools" described in this manual indicates applications which run on Windows.

"Messages" described in this manual indicates messages transferred between tools corresponding to PowerFRAMEVIEW.

"Script engine" described in this manual indicates the interpreter which executes and interprets the script.   "Engine" in this manual works like the "script engine."

Documents Used by Underlined Texts

Coding

- Items enclosed in angle brackets "<" and ">" are optional.
- "..." (for iteration) indicates that the preceding items enclosed in angle brackets (<and>) are described repeatedly.
- Items can be described in English or Japanese.   You can use either language.

Constants

Constants are values or strings available for use in the script.

Parentheses Expression

An enclosed expression is an expression enclosed in parentheses.

Public Variable

A public variable is a variable defined in a public paragraph.   This variable is valid as long as an object exists.   It can be referred to by the other objects.

Private Variable

A private variable is a variable defined in a private paragraph.   This variable is valid as long as an object exists.   It can be referred to by the other objects.

Local Variable

A local variable is a variable defined in a block.

Full-set Language Specification

```
public : Variable definition;
private : Variable definition;
Function() : Variable definition;
{
Variable definition;
Statement;
...
}
...
```

Sub-set Language Specification

Variable definition;

Statement;

Debugger

The debugger is a tool which investigates the cause of a malfunction of a script (if any), corrects it, and reconfirms the repaired script.

Return Value

A return value is a value to be posted when a function is executed successfully.   A null type constant (null) is posted when a return value does not exist.   When a function is not executed normally, a null type constant (null) is always posted.

List

A list is a group of strings which are separated from each other by a delimiter.

Example)

"ABC,DEF,GHI,....."	Delimited by ","
"ABC, DEF, GHI, ..."	Delimited by ",_"

Item

Items are character strings constituting a list.   Items are numbered in the order of 0, 1, 2, 3, ...

Example)   In the list "ABC.DEF.GHI,...," "ABC" is the 0th item.

PowerFRAMEVIEW Environment Setting

This function assembles an application so that it can be used by PowerFRAMEVIEW and sets information of menus of the PowerFRAMEVIEW developing manager.

To use this function, select "Environment Set..." from the OPTION menu of the PowerFRAMEVIEW developing manager.

START Message

This message is transferred to start a tool in the PowerFRAMEVIEW developing manager.

This message is transferred in the following cases:

- Starting a tool for PowerFRAMEVIEW
- Posting that a tool has been started successfully
- Posting that a tool has not been started successfully

Macro Manager

The macro manager executes a script describing a process corresponding to a message and controls transfer of a message between PowerFRAMEVIEW and a corresponding tool.

Modal Dialog Box

This dialog box is used to halt the execution of a program until the user responds to the displayed dialog box.

Modeless Dialog Box

This dialog box is displayed when the user must transfer information in succession to a program in progress.

Tool Class

PowerFRAMEVIEW employing the concept of tool classes assigns an identical tool class to tools of the same type.

Define a tool to be started according to a combination of a file extension and a tool class name by the Set PowerFRAMEVIEW Environment function and start the tool by specifying the tool class name and the name of the file to be processed by the tool.

Originator

An originator is a character string indicating a message source ID.

Start Tool

A start tool is a tool started by by ToolExec or ToolExecEX.

Descendant Tools

This tool is started by a Start Tool.

Labeled Parameter

This parameter can specify an argument of a function by the name when a function is defined or called.

The use of labeled parameters enables transfer of parameters without considering the order of the description of arguments.

Example)

Definition of function

```
func(label1 : para1, label2 : para2)   {   ...   }
```

Invocation of function

```
var   data1   =   1   ;
var   data2   =   2   ;
//Invocation of function
//para1 argument of function <- data1 value
//para2 argument of function <- data2 value
func(label2 : data2, label1 : data1)   ;
```

Positional Parameter

This parameter has a significant order of description of arguments of a function when the function is defined or called. The order of arguments when a function is defined must be equal to the order of arguments when the function is called.

Example)

Definition of function

```
func(para1, para2)   {   ...   }
```

Invocation of function

```
var   data2   =   2   ;
var   data1   =   1   ;
//Invocation of function
//para1 argument of function <- data1 value
//para2 argument of function <- data2 value
```

Win32

This explains remarks (restrictions, notices, etc.) of the Win32 version.

Introduction Of Script

This explains a script.

Roles and functions of a script

Script Coding

This explains how a script is described.

Use Of Script

This explains how a script is used.

Use of script

Creating and Updating Scripts

Execution Tools Installed in the PowerFRAMEVIEW

Particular functions

Callback Function

Exec Function

Terminate Function

Script debugger

When the execution of a script fails, use the Debugger to correct the script.

Debugger Function

Starting the Debugger

Script Functions

This explains the functions of a script.

The script functions are to get a date, to display a simple dialog box, etc.

The explanation of each function contains a description of the invocation type (Win16 or Win32) of a function in its title.

- Invocation by Win16 only ---> Described as "[Win16]"
- Invocation by Win32 only ---> Described as "[Win32]"
- Invocation by Win16 and Win32 ---> No description

Coding Example of Message Transmission

Coding examples for sending or receiving a message with the tool corresponding to PowerFRAMEVIEW are shown below.

See Script Example for Installing a Tool in the PowerFrameview for details.

```
Exec ()
      var Filespec ;
      MessageOpen(*EDIT*) ;
      Filespec = MakeFilespec("*", 0) ;
      MakeEvent("R", *EDIT*, "START", Filespec, "start") ;
      return (TRUE) ;

start ()
      var Get_MessageID_= GetMessageID();
      var Make_MessageID_= MakeMessageID();
      var Filespec_= GetFilespec();
      var SenderToolClass_= GetSenderToolClass() ;
      var MessageType_= GetMessageType() ;
      var ToolClassName_= GetToolClass() ;
      var CommandName_= GetCommand() ;
      var Data_= GetData(1) ;
      var Originator_= GetOriginator();
      var PathName_= FilespecToPathName(Filespec) ;
      MakeEvent("R", *EDIT*, "STOP", Filespec, "stop") ;
      ...
      MessageSend(Get_MessageID, "N", EDIT, "START", Filespec) ;
      return (TRUE) ;

stop ()
      ...
      ...
      DeleteEvent("R", *EDIT*, "START" , Filespec) ;
      DeleteEvent("R", *EDIT*, "STOP"   , Filespec) ;
      MessageClose();
      Close();
      return (TRUE) ;

Terminate ()
      ...
```

Message Open

<<Function>>   Opens the message path with the message manager.   Make sure to call this function before calling other Encapsulate functions.

<<Coding>>   MessageOpen(class)

<<Argument>>   Class = Specify the character string indicating your own Tool Class.

<<Return Value>>   The character string specified with the class is reported for normal end. null is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

The Message Close

<<Function>>   Closes the message path with the message manager.   Make sure to call this function before terminating a tool.   Other Encapsulate Functions must not be called after this function is called.

This function closes the message path opened by calling the MessageOpen coded in the same script.   Sending and receiving messages for the Tool Class specified with the MessageOpen (opening the path) are stopped by calling this function.

<<Coding>>   MessageClose()

<<Argument>>   None

<<Return Value>>   The character string indicating the Tool Class name is reported for normal end.   null is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Make Event

<<Function>>   Registers the message pattern received by the tool for the message manager.   Up to 500 messages can be registered.

<<Coding>>   MessageEvent(type,class,command,   specification,callback)

<<Argument>>

Type = Specify the character string indicating the type of the message to be received.

"R":   Request message

"N":   Report message (simple reporting or succeeded request)

"F":   Report message (failed request)

Class = Specify the character string indicating the Tool Class name of the message to be received.

Command = Specify the character string indicating the command name of the message to be received.

Specification =   Specify the character string indicating the file specification of the message to be received.

Callback = Specify the character string indicating the Callback Function called when a message is received.   Code the function having the same name as this parameter name in the script.

<<Return Value>>   The character string specified with the command is reported for normal end.   null is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

<<Remarks>>   "*" and "-" can be specified for the argument.

The meanings of "*" and "-" specifications are as follows:

"*":   The argument with this specification is not used as the parameter of the corresponding message.

"-":   The argument with this specification uses the default prepared with the receive tool of the corresponding message.

Message Send

<<Function>> Sends a message. This function can be called independently regardless of other Encapsulate Functions.

<<Coding>> MessageSend(ID, type, class, command specification,<,data...>)

<<Argument>>

ID = Specify the character string indicating the message ID. The ID must be specified as follows:

- Reply for the received message

Message ID acquired with the GetMessageID

- Sending a message other than the above (such as an independent message)

Message ID acquired with the MakeMessageID

Type = Specify the character string indicating the type of the message to be sent.

"R": Request message

"N": Report message (simple reporting or succeeded request)

"F": Report message (failed request)

Class = Specify the character string indicating the Tool Class name of the sending destination.

Command = Specify the character string indicating the command name of the message.

Specification = Specify the character string indicating the file specification of the message.

Data = Specify the character string indicating data. Multiple specifications are possible (omit when data is not to be sent. Up to 10 items of data can be sent.

<<Return Value>> The character string specified with the command is reported for normal end. null is reported for abnormal end.

<<Example>> See Coding Example of Message Transmission.

<<Remarks>> "*" and "-" can be specified for the argument. Meanings of "*" and "-" specifications are as follows:

"*": The argument with this specification is inappropriate for the parameter of the corresponding message. Therefore, this argument of the corresponding message is not used.

"-": The argument with this specification uses the default prepared with the receive tool of the corresponding message.

Delete Event

<<Function>>   Deletes a message pattern registered for the message manager.

<<Coding>>   DeleteEvent(type, class, command, and specification)

<<Argument>>

Type = Specify the character string indicating the type of the message to be sent.

"R":   Request message

"N":   Report message (simple reporting or succeeded request)

"F":   Report message (failed request)

Class = Specify the character string indicating the Tool Class name of the message to be deleted

Command = Specify the character string indicating the command name of the message to be deleted

Specification =   Specify the character string indicating the file specification of the message to be deleted.

<<Return Value>>   The character string specified with the command is reported for normal end.   null is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Make Message ID

<<Function>>   Creates a message ID.

<<Coding>>   MakeMessageID()

<<Argument>>   None

<<Return Value>>   The character string indicating the message ID   is reported for normal end.   null is reported for abnormal end.

<<Example>>   See Coding Example of Message Ttransmission.

Make File spec

<<Function>>   Creates the file specification.

<<Coding>>   MakeFilespec(path and status)

<<Argument>>

Path = Specify the character string indicating the name of the path of which file specification is to be acquired.

State = Specify the status of a file with a number.

1:   Is not present.

2:   Is present.

0:   Whether a file is present is unknown.

<<Return Value>>   The character string indicating the file specification is reported for a normal end.   null is reported for an abnormal end.

<<Example>>   See Coding Example of Message Transmission.

File Spec To Path Name

<<Function>>   Acquires the absolute path name corresponding to the file specification.

<<Coding>>   FilespecToPathName(specification)

<<Argument>>   Specification = Specify the character string indicating the name of the absolute path of which file specification is to be acquired.

<<Return Value>>   The character string indicating the absolute path name is reported for normal end.   null is reported for abnormal end.

<<Example>>   See Coding Example of Message Ttransmission.

Get Sender Tool Class

<<Function>>   Acquires the tool class name of the message sending source.

<<Coding>>   GetSenderToolClass()

<<Argument>>   None

<<Return Value>>   The character string indicating the Tool Class name in the sending source is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get Message ID

<<Function>>   Acquires the message ID.

<<Coding>>   GetMessageID()

<<Argument>>   None

<<Return Value>>   The character string indicating the message ID is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get Message Type

<<Function>>   Acquires the message type.

<<Coding>>   GetMessageType()

<<Argument>>   None

<<Return Value>>   The character string indicating the message type is reported for normal end.

"R":   Request message

"N":   Report message (simple reporting or succeeded request)

"F":   Report message (failed request)

A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get Tool Class

<<Function>>   Acquires the Tool Class name of the message.

<<Coding>>   GetToolClass()

<<Argument>>   None

<<Return Value>>   The character string indicating the tool class name is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get Command

<<Function>>   Acquires the command name of the sending message.

<<Coding>>   GetCommand()

<<Argument>>   None

<<Return Value>>   The character string indicating the command name is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get File Spec

<<Function>>   Acquires the file specification of the message.

<<Coding>>   GetFilespec()

<<Argument>>   None

<<Return Value>>   The character string indicating the file specification is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

Get Data

<<Function>>   Acquires message data.

<<Coding>>   GetData(sequence)

<<Argument>>   Sequence = Specify the number indicating the sequence of data.   Specify 1 for the first data.   Numbers 1 to 10 can be specified.

<<Return Value>>   The character string indicating the data of the specified message sequence is reported for normal end.   A null character string is reported if data does not exist.

<<Example>>   See Coding Example of Message Transmission.

Get Originator

<<Function>>   Acquires the Originator of the message.

<<Coding>>   GetOriginator()

<<Argument>>   None

<<Return Value>> The character string indicating the originator of the message is reported for normal end.   A null character string is reported for abnormal end.

<<Example>>   See Coding Example of Message Transmission.

ToolExec

<<Function>>   Starts the tool (Win16 application).   For starting the Win32 application, see ToolExecEX.

<<Coding>>   ToolExec(tool<, <parameter><, flag <, <function><,descendants<,descendants...>>>>>)

<<Argument>>

Tool = Specify the character string indicating the file name of the Starting Tool.   Specify the file name only.

Parameter = Specify the character string indicating the starting option of the starting tool. The string can be up to 118 bytes.   (optional)

Flag = Specify how to monitor the tool.

TRUE: Monitors the tool.   Performs the following processing when the starting tool and all Descendant Tools are terminated.

- If a script is not being executed, executes the Terminate function immediately.
- If a script is being executed, interrupts the processing and executes the Terminate function.

FALSE: Monitors the tool.   Performs the following processing when the starting tool and all descendant tools are terminated.

- If a script is not being executed, executes the Terminate function immediately.
- If a script is being executed, executes the Terminate function after the current processing terminates.

Omission: Does not monitor the tool.   Performs no operation after the starting tool is terminated.   Specified values other than the above are assumed to be omitted.

Function = Specify the character-string type function name of a script to be executed when the starting tool or descendant tool is terminated.   The Terminate Function is assumed to be specified if omitted.   Make sure to code Close in this function.

Descendants = Specify the character-string type file name of the descendant tool which monitors termination.   If the file name is specified without a full-path name, all descendant tools having the same file name are monitored regardless of the path.   Up to 16 descendant path names can be specified.   Specifying tools having the same file name may cause an error even if the number of descendant tool names is 16 or less.   (Specify up to 256 system starting tools and descendant tool tasks in total.)

<<Return Value>>   The object ID of the started tool is reported for a normal end.   null is reported for abnormal end.

<<Example>>   An example of monitoring termination of descendant tools "Tool1.exe" and "c:\dir\Tool2.exe" by starting starting tool "tool.exe" is shown below.   In this example, when the starting tool and two descendant tools are all terminated, the Term1 function is called. The Term1 function must be coded in the same script source.

```
public :
var tool ;
tool = ToolExec("tool.exe", "-a -b -c file.c", TRUE, "Term1",
                        "Tool1.exe", "c:\\dir\\Tool2.exe") ;
...

Term1() {
}
```

See Character-string Constant for how to write symbol '\'.

Scp Open

<<Function>>   Starts the Macro Manager in Full-set Language Specification, and registers a script coded with the full-set language specification.   When this function has terminated normally, a function in a script can be executed using the return value.

When a function in the script is to be executed, code the function name after the object ID. Only invalid-type, numeric type, and character-string type arguments can be transferred to a function in a script.   The values receivable as function return values are also limited to these same types only.   To use multiple scripts with the full-set language specification, use this function to register them for each script.

<<Coding>>   ScpOpen(file)

<<Argument>>   File = Specify the character string indicating the script name coded with the full-set language specification.   If the path is omitted, the file conforms to the path information.

<<Return Value>>   The object ID is reported for normal end.   null is reported for abnormal end.

<<Example>>   Script in the calling source

```
var o ;
var ret ;

o = ScpOpen("script.scp") ;

ret = o.func(123) ;



closetask(o) ;
```

Script in the calling destination

```
func(Para1) {
var i ;
i =   Para1 * 10 ;
return (i) ;
}
func2(Para1, Para2) {
}
```

Scs Open

<<Function>>   Starts the Macro Manager in the Subset Language Specification.   When this function has terminated normally, a function in a script coded with the subset language specification can be executed using the return value.

When a script is to be executed, code the script name after the object ID.   At this time, the "." of the file name extension must be escaped.

Only invalid-type, numeric type, and character-string type arguments can be transferred to a script with the subset language specification.   The values receivable as function return values are also limited to these same types only.   To receive an argument in a script with the subset language specification, the Getparameter (receiving the parameter) must be called.

For using multiple scripts with the subset language specification, call this function only once.

<<Coding>>   ScsOpen()

<<Argument>>   None

<<Return Value>>   The object ID is reported for normal end.   null is reported for abnormal end.

<<Example>>   Script in the calling source

```
var o ;
var ret ;
o = ScsOpen() ;
ret = o.script\.scs(123) ;

closetask(o) ;
```

Script in the calling destination

```
var   arg, size, i ;
arg = GetParameter() ;
size = NumberArray(arg) ;

i =   arg[1] * 10 ;
return (i) ;
```

In a script in the calling destination, the parameter specified at script execution can be received by calling the Getparameter.   To know the number of received parameters, call the NumberArray.

Set Failure Reply

<<Function>>   Sets how to correspond the message received by the macro manager during script engine execution.

The FALSE mode becomes effective immediately after the message path is opened.   A message received while the script engine is not in execution is processed regardless of the setting with this function.

The macro manager executes the processing using the script engine when it receives a message.   Therefore, subsequent messages can be processed only when processing of the previous message has terminated.   If a message is received during script engine execution, the message is not processed, and the message is requested to be resent from the sending source.   However, this causes permanent message resending.   To make the macro manager forcibly reply with the subsequent message as a failure to the sending source, use this function.

<<Coding>>   SetFailureReply(mode)

<<Argument>>

Mode = Specify how to correspond the message.

TRUE: Sends the "failure" reply message for the received message to the tool class in the sending source if a message is received.

FALSE: Requests the message manager to resend the received message if a message is received.

<<Return Value>>   The number indicating the corresponding method is reported.   (TRUE: FALSE)

<<Example>>   SetFailureReply(TRUE);

Tool Exec EX

<<Function>>   Starts the tool (Win32 application).   For starting the Win16 application, see ToolExecEX.

<<Coding>>   ToolExec(tool<, <parameter><, flag <, <function><,descendants>>>>>)

Starting tool32(tool<, <parameter><, flag <, <function><,descendants>>>>>)

<<Argument>>

Tool = Specify the character string indicating the file name of the starting tool.   Specify the file name only.

Parameter = Specify the character string indicating the starting option of the starting tool. The string can be up to 118 bytes.   (optional)

Flag = Specify how to monitor the tool.

TRUE: Monitors the tool.   Performs the following processing when the starting tool and all Descendant tools are terminated.

- If a script is not being executed, executes the Terminate function immediately.
- If a script is being executed, interrupts the processing being executed and executes the Terminate function.

FALSE: Monitors the tool.   Performs the following processing when the starting tool and all descendant tools are terminated.

- If a script is not being executed, executes the Terminate function immediately.
- If a script is being executed, executes the Terminate function after the current processing terminates.

Omission: Does not monitor the tool.   Performs no operation after the starting tool is terminated.   Specified values other than the above are assumed to be omitted.

Function = Specify the character-string type function name of a script to be executed when the starting tool or descendant tool is terminated.   The Terminate Function is assumed to be specified if omitted.   Make sure to code Close in this function.

Descendants = Specify whether to monitor termination of descendant tools.

TRUE: Monitors all descendant tools.   Monitors the starting tool and all descendant tools.

FALSE: Does not monitor descendant tools.   Monitors only the starting tools.

<<Return Value>>   The object ID of the started tool is reported for a normal end.   null is reported for an abnormal end.

<<Example>>   An example of monitoring termination of descendant tools by starting starting tool "tool.exe" is shown below.   In this example, when the starting tool and all descendant tools are terminated, the Term1 function is called.   The Term1 function must be coded in the same script source.

```
public :
var tool ;
tool = ToolExecEX("tool.exe", "-a -b -c file.c", TRUE, "Term1",
                        TRUE ) ;
...

Term1() {
}
```