J*o*PPS - Script

JoPPS Windows 3.38 – mei 2021

J*o*PPS-Script / J*o*PPS Windows v3.38
TechWIN Software BVBA - 2018
Brusselsesteenweg 267, B-2800 Mechelen.

Contents

Introduction

This document is intended for advanced users. Its purpose is to bring you a brief overview of the functionality of J*o*PPS-Script.

Before you start reading this document you should :

- Master Microsoft Windows and know how to work with files and folders.
- Know how to use J*o*PPS, be familiar with its concepts and terminology.
- Have a basic programming knowledge or understand the concepts involved.

We hope this document is easy-to-read yet complete enough to answer all your questions related to J*o*PPS-Script.

Koen Verbeeck
TechWIN Software BVBA
July 2000.

1. J*o*PPS-Script

J*o*PPS-Script is an interpreted scripting language designed to automate simple yet repetitive tasks
within the J*o*PPS program.

Beside its power to code handy macro's it can be used to interface J*o*PPS with other
software packages such as MS Word through OLE automation and ActiveX.

Use *JScripter* (see 5. Writing scripts using JScripter) to explore and learn J*o*PPS-Script without the need of J*o*PPS.
JScripter is a simple to use interactive programming environment designed to code and test
J*o*PPS-Script programs. It offers a variable watch window and step-by-step execution.

You can run J*o*PPS-Scripts from the command line or from your batch files using *JCall*;
a lightweight command line J*o*PPS-Script interpreter.
(see 6. Running J*o*PPS-Scripts from the command line: JCALL)

JScripter and JCall support the same set of basic functions, J*o*PPS however adds a lot of
extra functions (and constants) to interface its UI. (see 3 Using J*o*PPS-Script in J*o*PPS)

The interpreter

The J*o*PPS-Script interpreter works in two phases:

First it starts with a *lexical analysis* of the entire script source, you could say it "compiles" the script
and prepares the execution phase.

Second, after the lexical analysis, the interpreter starts executing the statements making up the script.
It starts at the first statement and works its way down until it has executed the last statement.

The execution can prematurely be interrupted by:

A run-time error
An `Halt` or `Fatal` statement (aborting the execution)
The interpreter being paused (by the user)

2. The language

A J*o*PPS-Script is made up out of statements; a statement is usually a function call or a variable assignment.

Statements are separated by semicolons ';'.

A statement block (or compound statement) is a group of related statements;
the entire block must be enclosed between braces ({ }) if it contains multiple statements.

2.1. Variables

In J*o*PPS-Script variables are declared implicitly; storage space is dynamically allocated the first time a variable is used.

There are no real datatypes in J*o*PPS-Script; a variable can hold any type of fundamental data: numeric (integer, floating points, boolean) , alphanumeric (string) and even represent instances of objects. (IDispatch or J*o*PPS-Script objects)

J*o*PPS-Script assigns a sub*type* to each variable during script execution. This subtype can change dynamically each time a new value is assigned to the variable:

```
a := 10;              /* variable a is of type numeric */
a := "JoPPS";         /* variable a is of type string */
```

Although variables can contain any type of data it will often be required to check for the type of data that is stored in a variable to ensure proper script execution.

Datatypes are implied by the use of variables in expressions or by passing variables as parameters to J*o*PPS-Script functions.

<u>We can divide these variable types into 4 categories:</u>

Variable		type
❶ holds a numeric value	floating point, integer, boolean (B-type), date-time value	D-type
❷ holds a alphanumeric value	string or character value	S-type
❸ holds an object instance	object instance	O-type
❹ holds an IDispatch interface	IDispatch object instance	I-type

We can verify the type of a variable using the following functions :

```
IF IsNumber(a) THEN OutputMsg('Is a number');
IF IsString(a) THEN OutputMsg('Is a string');
IF IsObject(a) THEN OutputMsg('Is an object');
IF IsIDispatch(a) THEN OutputMsg('Is an IDispatch interface');

/*for objects we can test for a NILL value..*/
IF IsObject(a) && IsNill(a) THEN OutputMsg('The object is NILL');
```

String values (S-type) are <u>always</u> between single (') or double (") quotes.

Boolean values (B-type) can be represented using D-type variables:

```
a := 10;       /* <> 0 = true */
IF a THEN OutputMsg("a is TRUE") ELSE OutputMsg("a is FALSE");
a := 0; /* = 0 = false */
IF a THEN OutputMsg("a is TRUE") ELSE OutputMsg("a is FALSE");
```

The language constants `TRUE` (=1) and `FALSE` (=0) can be used instead of 0 and <>0 to represent the boolean values true and false.

```
a := TRUE;
IF a THEN OutputMsg("a is TRUE") ELSE OutputMsg("a is FALSE");
a := FALSE;
IF a THEN OutputMsg("a is TRUE") ELSE OutputMsg("a is FALSE");
```

Note that the language constant TRUE only represents the value 1, this is not the same as "TRUE" which is actually not zero (<>0).

A variable can be deallocated (freed) by means of the `Kill` function.

```
Kill("a");     /* the argument is the name of the variable to free */
```

At the moment J*o*PPS-Script has no array type.

<u>Variable naming rules:</u>

- A variable name must begin with an alphabetic character or underscore ("_").
  The remainder of the name may contain any alphanumeric characters including underscores.
- The length of the variable is not limited but should be kept as short as possible to improve readability.
- Periods may not be embeded in a variable name.
- A variable name must be unique.
- Variable names are not case-sensitive. (as J*o*PPS-Script is not case-sensitive)

<u>Variable assignments:</u>

To assign a value to a variable use the assignment operator (e.g. `:=`) in the format <variable> := <value>.

```
a := 10;
b := "JoPPS is great";
c := TRUE;
e := d := 5.5;
IF (f := (a + d)) THEN OutputMsg(NumToStr(f));
g := Strings.Create();
```

2.2. Operators

J*o*PPS-Script provides all standard operators that you would expect in a development environment:

Arithmetic operators		Example (`b := 10`, `c := 3`)	
`+`	addition	`a := b + c`	`13`
`-`	subtraction	`a := b - c`	`7`
`*`	multiplication	`a := b * c`	`30`
`/`	division	`a := b / c`	`3.33333333333333`
`//`	integer division (DIV)	`a := b // c`	`3`
`\`	modulo	`a := b \ c`	`1`
`\\`	integer modulo	`a := b \\ c`	`1`
`**`	power	`a := b ** c`	`1000`
`&&`	logical and	`a := b && c`	`TRUE`
`&`	bitwise and	`a := b & c`	`2`
`||`	logical or	`a := b || c`	`TRUE`
`|`	bitwise or	`a := b | c`	`11`
`^^`	logical xor	`a := b ^^ c`	`FALSE`
`^`	bitwise xor	`a := b ^ c`	`9`
`=`	equality	`a := b = c`	`FALSE`
`<> or !=`	inequality	`a := b <> c`	`TRUE`
`!`	logical negation	`a := !(b = c)`	`TRUE`
`~`	bitwise negation	`a := ~(b = c)`	`-1`
`>`	greater than	`a := b > c`	`TRUE`
`<`	less than	`a := b < c`	`FALSE`
`<=`	less than or equal to	`a := b <= c`	`FALSE`
`>=`	greater than or equal to	`a := b >= c`	`TRUE`
String operators		**Example (`e := "JoPPS-Script is "`, `f :="great"`)**	
`+`	concatenation	`d := e + f`	`"JoPPS-Script is great"`
`=`	equality	`d := e = f`	`FALSE`
`<> or !=`	negation	`d := e <> f`	`TRUE`
`>`	greater than	`d := e > f`	`FALSE`
`<`	less than	`d := e < f`	`TRUE`
`<=`	less than or equal to	`d := e <= f`	`TRUE`
`>=`	greater than or equal to	`d := e >= f`	`FALSE`

Binary operators operate on two operands, unary operators operate on only one operand.

The + and - operators can be used as binary or as unary operator.
The ! (negation) and the ~ (bitwise negation) operator are strict unary operators.

Bitwise operations are always performed 32-bit, results are threated as signed 32-bit integers.

2.3. Predefined language constants

You can use the following constants in expressions in J*o*PPS, JScripter or JCALL :

Constant	Value
`TRUE`	1 or logical true, note that all numeric values <> 0 are actually TRUE
`FALSE`	0 or logical false
`CRLF`	String representing a CR/LF pair. (ascii characters 13 and 10) Can be used to write textlines to a DOS ascii file.
`NIL`	Represents the value of a non-allocated object variable.

Note: constants specific to J*o*PPS itself are discussed in section *9. Function reference*.

2.4. Expressions

Evaluation rules :

- Unary operators and assignments are evaluated from right to left.
- Binary operators are evaluated according to their priority and from left to right.

Binary operators in ascending order	
`||  ^^`	Lowest priority
`&&`	
`=  !=  <>  >  >=  <  <=`	
`+  -  |  ^`	
`*  /  //  \  \\  &`	
`**`	Highest priority

- Logical expressions are evaluated completely,
  no lazy (or short-circuit) evaluation is used !

Expression examples :

`a := 10`	Expression is a simple assignment, a is assigned the value 10.
`a := b := c := 10`	a, b and c are assigned the value 10. The variable c is assigned first, b second and the variable a last.
`a+1 < 10`	The priority of the + operator exceeds the priority of the < operator thus the sub-expression a+1 is evaluated first. If the variable a has a value equal to or greather than 9 before the expression is evaluated the result of the expression will be `FALSE`.
`1+2*3`	The priority of the * operator exceeds the priority of the + operator thus the result will be 7.
`(1+2)*3`	The sub-expression between the parentheses is evaluated first so that the result will be 9.
`FALSE && (a:=a+1)`	The value of the variable a is incremented although the first sub-expression is `FALSE`. No lazy evaluation is applied as it is used in some other languages. (e.g. Pascal)

`a=0 && b>10 || a!=0 && b<-10`
is similar to
`((a = 0) && (b > 10)) || ((a != 0) && (b < -10))`	

`a := -b := 1`
is invalid and should be
`a := -(b := 1)`
	The first expression is not allowed ! The assignment operator := cannot assign a value to an expression.

`!(a && b)`
is similar to
`!a || !b`	

2.5. Language structures

2.5.1. Conditional execution : IF - THEN – ELSE

Like other languages, J*o*PPS-Script includes the ability of conditional execution by means of the `IF-THEN-ELSE` statement:

Example :

```
/* start word */
word := START("word.application");

/* word up and running? */
IF IsIDispatch(word) THEN
{
  /* Yes it is */
  word.Visible := TRUE;
  doc := word.Documents.Add();
  selection := word.ActiveDocument.ActiveWindow.Selection;
  selection.TypeText("Hello from JoPPS-Script");
}
ELSE
{
  /* Oops, it is not */
  Beep();
  Fatal("Sorry, I was unable to start WORD");
};
```

The general syntax for the `IF-THEN-ELSE` statement looks like this:

```
IF <expr> THEN
  statement block ❶
[ ELSE
  statement block  ❷   (optional)
]
```

Statement block ❶ is executed when the `IF`-expression evaluates `TRUE`,
otherwise (when the `IF`-expression evaluates `FALSE`) optional statement block ❷ is executed.

2.5.2. Iteration : WHILE - DO - BREAK - CONTINUE

Iterations can be programmed using a `WHILE` - `DO` statement.

Example :
```
/* count till 10 */
i := 0;
WHILE (i:=i+1) <= 10 DO OutputMsg("i is "+IntToStr(i));
```

The syntax for the `WHILE-DO` statement looks like this:

```
...
WHILE <expr> DO
  statement block ❶
...
```

Statement block ❶ is executed as long as <expr> evaluates `TRUE`,
otherwise execution continuous at the first statement following the statement block.

When a statement block consists of more then one statement the entire block must be enclosed between braces (`{ . . . }`).

BREAK

`BREAK` causes the flow of control to exit the current `WHILE` statement.

This will cause script execution to continue at the first statement following statement block ❶.

We could rewrite our previous example using `BREAK` :

```
/* count till 10 */
i := 1;
WHILE TRUE DO
{
  OutputMsg("i is "+IntToStr(i));
  IF i = 10 THEN BREAK;
        i := i + 1;
};
```

CONTINUE

`CONTINUE` allows the flow of control to proceed to the next iteration of the `WHILE` statement.

Example rewritten with `CONTINUE :`
```
/* count till 10 */
i := 1;
WHILE TRUE DO
{
  OutputMsg("i is "+IntToStr(i));
  i := i + 1;
  IF i <= 10 THEN CONTINUE;
  BREAK;
};
```

Warning : infinitive loops can lockup your computer - be carefull.

Note : J*o*PPS-Script has no other iteration statements such as a `FOR` loop or a `REPEAT-UNTIL`.

2.5.3. Jumping: Labels and GOTO

The `GOTO` statement can be used to continue the script execution at a specific location somewhere else in the same script: make a jump.

The locations we can jump to are marked with *labels*.

A goto statement is followed by a label identifier defining the location where to jump to..

Example:
```
errmsg := "";  /* no error */
word := START("word.application");
IF !IsIDispatch(word) THEN
{
  errmsg := "Sorry, I was unable to start WORD";
  GOTO error;  /* make jump to errorhandler */
};
word.Visible := TRUE;
doc := word.Documents.Add();
selection := word.ActiveDocument.ActiveWindow.Selection;
selection.TypeText("Hello from JoPPS-Script");

@error:
IF errmsg <> "" THEN /* error? */
{
  Beep();
  Fatal(errmsg); /* script execution stops here */
};
OutputMsg("Program terminated");
```

The syntax for the `GOTO` statement looks like this:

```
..
...
GOTO <labelname ❷>
...
..
@<labelname ❶>:
...
..
@<labelname ❷>:
..
...
GOTO <labelname ❶>
...
..
```

Label identifiers follow the same naming rules as variable identifiers.
A label definition consists of a label identifier between a "@" and a ":" character.

The "@" character tells the interpreter it has to deal with a label definition instead of a regular statement.

2.6. Stopping script execution : Halt - Fatal

You can terminate a script being executed by calling `Halt`, execution will stop immediately <u>without</u> raising an error.

```
word := START("word.application");
IF !IsIDispatch(word) THEN
{
  MsgErr("WORD did not start !");
  Halt;
};
```

On the other hand you can abort a script and raise an error by calling `Fatal`.
Execution will stop immediately as with `Halt` but an error is triggered.
You can pass an error message to inform the end-user about the kind of error occurred.

```
word := START("word.application");
IF !IsIDispatch(word) THEN Fatal("WORD did not start !");
```

2.7. Objects

With J*o*PPS v2 J*o*PPS-Script offers the ability to instanciate object variables that can be used to perform various tasks such as writing to a diskfile or accessing the Windows registry.

It is not possible, nor is it intended, to create new object classes - or to derive your own object classes.

The classes provided in J*o*PPS-Script are only intended to be used in your scripts: they group methods and properties logically belonging together into a single variable.

2.7.1. Object classes

The standard v2 syntax comes with the following predefined object classes :

STRINGS class	Handles lists of strings (eg. texts)
REGISTRY class	Interfaces the windows registry
INIFILE class	Interfaces windows INI-files
HANDLE class	Interfaces windows filehandles (e.g. a diskfile or a named pipe)

Refer to *2.7.3. Predefined object classes* for a detailed discussion on these object classes.

In J*o*PPS itself various object classes are added to the list :
to interface the JoPPS database
to work with projects and projectdata
to interface the modellibrary
to program simple dialogs to interact with the user
to interface machining centers

Fore more information about working with objects in J*o*PPS refer to *3.5 JoPPS related objects*.

Simple dialogs can be created using the Form Objects. Refer to *4. Talking to the user : Form objects* for more information about creating dialogs.

2.7.2. Object variables : instanciating and destroying

To use an object start with instanciating an object variable : allocating memory to hold the object and initialize its inner workings.

This can be done by calling an object's **constructor** Create.

```
oText := Strings.Create(); /* a constructor is a method so it needs () */
```

After the call the variabele oText holds an instanciated STRINGS object. If memory allocation fails the variabele oText will be Nil and cannot be used. We can test for a Nil object using the function **IsNil**:

```
if IsNil(oText) then Halt; /* Oops, something went wrong here.. */
```

or by

```
if oText = Nil then Halt;
```

Note that IsNil can also be used on non-object variables.

Calling methods or addressing properties of a nil object will cause a run-time error.

Important:
Do not instanciate objects of classes where the classname begins with and underscore character !
These classes are for internal use only, they can usually be accessed by means of a system variable.

We are now ready to use our newly instanciated object; by first testing for a Nil value we made sure we are now dealing with a valid object.. We are ready to use its **properties** and call its **methods**:

```
if oText.LoadFromFile("c:\autoexec.bat") then
  ShowMessage("My autoexec.bat file contains "+IntToStr(oText.Count)+" lines of text")
else
  ShowMessage("You have no autoexec.bat !");
```

We can assign an object variable (O-type variables) like any other variable:

```
oText2 := oText;
```

Now the variable oText2 points to the same object as oText does; objects are in fact pointers so assigning object variables just DUPLICATES THE REFERENCE, NOT THE OBJECT!

We can now use oText2 in the same way we use oText. Both variables reference the same object instance.

Instanciated objects occupy dynamic memory and should be freed when no longer needed. Free an object by calling its **destructor** Free:

```
oText.Free(); /* a destructor is a method so it needs () */
```

Once the object is freed the variable oText (and oText2) will be Nil.

Using a Nil object will trigger a runtime error. Executing the next statement raises a runtime error as oText2 is Nil (because we freed oText and oText2 pointed to the same object instance) :

```
oText2.SaveToFile("c:\autoexec.bak"); /* This will give a runtime error */
```

Freeing an object garantees any windows resources (e.g. a filehandle) allocated by the object are properly released. Normally object instances are freed automatically by J*o*PPS-Script when your script terminates but it is good policy to free all objects you instanciate.

2.7.3. Predefined object classes

The V2 syntax comes with a number of build-in object classes. You can uses these classes in JScripter or J*o*PPS.

2.7.3.1. The ARRAY class

To be documented

Overview

Constructor	
`CREATE (Dfh)`	
Destructor	
`FREE ()`	
Methods	
`ADD ( ? ) : ?`	
`COPYFROM ( ? ) : ?`	
`INIT ( ? ) : ?`	
`REDIM ( ? ) : ?`	
Properties	
`HIGH`	
`LENGTH`	
`LOW`	

2.7.3.2. The INIFILE class

The INIFILE class implements an interface to Windows *.INI files.

Example :
```
ini := IniFile.Create("c:\joppswin\jopps.ini");
language := ini.ReadInt("Language","Language",0);
msg := "The default JoPPS language is ";
IF language = 0 THEN msg := msg + "Dutch"
ELSE IF language = 1 THEN msg := msg + "French"
ELSE IF language = 2 THEN msg := msg + "German"
ELSE IF language = 3 THEN msg := msg + "English"
ELSE msg := msg + "undefined !";
ini.Free();
```

Overview

Constructor	
`CREATE (Sfn)`	Returns an instanciated inifile object. Sfn is the name of the INI file to be used
Destructor	
`FREE ()`	Frees the INI file object
Methods	
`READSTRING (Ssection,Skey,Sdef) : S`	ReadString reads a string value from an INI file. Ssection identifies the section in the file that contains the desired key. Skey is the name of the key from which to retrieve the value. Sdef is the string value to return if the: -section does not exist -key does not exist -data value for the key is not assigned
`WRITESTRING (Ssection,Skey,S)`	WriteString writes a string value S to an INI file. Ssection identifies the desired section in the file, Skey is the name of the key.
`READINT (Ssection,Skey,Ddef) : Dint`	ReadInt reads an integer value from an INI file. Ssection identifies the section in the file that contains the desired key. Skey is the name of the key from which to retrieve the value. Ddef is the value to

	return if the: -section does not exist. -key does not exist. -data value for the key is not assigned.
`WRITEINT (Ssection,Skey,Dint)`	WriteInt writes an integer value Dint to an INI file. Ssection identifies the section in the file, Skey is the name of the key.
`READBOOL (Ssection,Skey,Ddef) : Dbool`	ReadBool reads a boolean value from an INI file. Ssection identifies the section in the file that contains the desired key. Skey is the name of the key from which to retrieve the value. Ddef is the boolean value to return if the: -section does not exist. -key does not exist. -data value for the key is not assigned.
`WRITEBOOL (Ssection,Skey,Dbool)`	WriteBool writes a boolean value to an INI file. Ssection identifies the section in the file, Skey is the name of the key.
`READDATETIME (Ssection,Skey,Ddef) : Ddatetime`	ReadDateTime reads a datetime value from an INI file. Ssection identifies the section in the file that contains the desired key. Skey is the name of the key from which to retrieve the value. Ddef is the datetime value to return if the: -section does not exist. -key does not exist. -data value for the key is not assigned.
`WRITEDATETIME (Ssection,Skey,Ddatetime)`	WriteDateTime writes a datetime value to an INI file. Ssection identifies the section in the file, Skey is the name of the key.
`READNUM (Ssection,Skey,Ddef) : D`	ReadNum reads a numeric value from an INI file. Ssection identifies the section in the file that contains the desired key. Skey is the name of the key from which to retrieve the value. Ddef is the numeric value to return if the: -section does not exist. -key does not exist. -data value for the key is not assigned.
`WRITENUM (Ssection,Skey,D)`	WriteNum writes a numeric value to an INI file. Ssection identifies the section in the file, Skey is the name of the key.
`SECTIONEXISTS (Ssection) : B`	Verifies if the specified Ssection section exists
`DELETESECTION (Ssection)`	Delete the specified section Ssection from the inifile
`KEYEXISTS (Skey) : B`	Verifies if the specified Skey key exists
`DELETEKEY (Skey)`	Delete the specified key Skey from the inifile
`UPDATEFILE ()`	Make sure changes made to the inifile are flushed to disk
Properties	
`FILENAME`	Returns the name of the inifile. (read-only)

2.7.3.3. The HANDLE class

The HANDLE class enables you to read from and write to communications resources identified by a Windows handle: a diskfile, a named pipe, etc.

Overview

Constructor	
`CREATE (Dfh)`	Returns an instanciated HANDLE object for a given handle. The handle must be obtained by opening or creating the resource in the appropriate mode. You can use `OpenFile` or `CreateFile` to obtain a valid handle to a diskfile.
Destructor	
`FREE ()`	Frees the handle object. It is up to you to release the handle passed to the constructor.
Methods	
`SEEK (Dpos)`	Moves the current position in the file to the specified offset Dofs.
`READ (SDI,Dbytes) : Dread`	Read Dbytes into the specified variable. The bytes are read starting at the current file pointer position. The file pointer is moved and the number of bytes actually read is returned.
`WRITE (SDI,Dbytes) : Dwritten`	Write Dbytes from the specified variable to the file. The bytes are writen starting at the current file pointer position. The file pointer is moved and the number of bytes actually written is returned.
Properties	
`HANDLE`	The handle used by the HANDLE object. (read-only)
`SIZE`	Returns the size in bytes of the data represented by the handle. (read-only)
`POSITION`	Returns the byte offset of the next byte to read or write. (read-write)

2.7.3.4. The PARAMETER class

Constructor	
CREATE (?)	
Destructor	
FREE ()	
Methods	
GETACTIONVAR(?) : ?	
Properties	
CODE	
FLAG	
NEIGHBOUR	
NEIGHBOURABS1	
NEIGHBOURABS2	
NEIGHBOURCOR1	
NEIGHBOURCOR2	
NEIGHBOURLENGTH	
NEIGHBOURMARK	
NEIGHBOUROFS1	
NEIGHBOUROFS2	
NEIGHBOURPOS1	
NEIGHBOURPOS2	
NEIGHBOURSIDE	
REF	
REFERENCE	
REFERENCEABS1	
REFERENCEABS2	
REFERENCECOR1	
REFERENCECOR2	
REFERENCELENGTH	
REFERENCEMARK	
REFERENCEOFS1	
REFERENCEOFS2	
REFERENCEPOS1	
REFERENCEPOS2	
REFERENCESIDE	
RESULT	
SELF	
SELFABS1	
SELFABS2	
SELFCOR1	
SELFCOR2	
SELFLENGTH	
SELFMARK	
SELFOFS1	
SELFOFS2	
SELFPOS1	
SELFPOS2	
SELFSIDE	
SRCCODE	
SRCDESC	
SRCID	
VALUE	
VICTIM	
VICTIMABS1	
VICTIMABS2	
VICTIMCOR1	
VICTIMCOR2	

VICTIMLENGTH	
VICTIMMARK	
VICTIMOFS1	
VICTIMOFS2	
VICTIMPOS1	
VICTIMPOS2	
VICTIMSIDE	

To be documented

2.7.3.5. The REGISTRY class

The REGISTRY class implements an interface to the Windows registry.

The windows registry is a system database that an applications can use to store and retrieve configuration information. It replaces the older *.INI configuration files used in Windows 3.x applications.
(see 2.7.3.2 *The INIFILE class*)

To be documented

Overview

Constructor	
CREATE (Dfh)	
Destructor	
FREE ()	
Methods	
ADD (?) : ?	
COPYFROM (?) : ?	
INIT (?) : ?	
REDIM (?) : ?	
Properties	
HIGH	
LENGTH	
LOW	

Configuration information is stored in a hierarchical tree. Each node in the tree is called a key. Every key can contain subkeys and data values that represent part of the configuration information for an application.

All keys that an application creates, opens, reads, or writes are subkeys of predefined root keys. By default, a REGISTRY object is created with a root key of HKEY_CURRENT_USER.

The REGISTRY class methods and properties are implemented to resemble their INIFILE counterparts where possible.

Example :
```
reg := Registry.Create("Software\TechWIN\JOPPS");
joppshomepath := reg.ReadString("START","InFolder","");
reg.Free();
ShowMessage("JoPPS is installed in folder <"+joppshomepath+">");
```

Overview

Constructor	
CREATE (Skey)	Returns an instanciated registry object
Destructor	
FREE ()	Frees the registry object
Methods	
DELETEKEY (Skey) : Dbool	Deletes the key identified by Skey.
DELETEVALUE (Skey,Svalue) : Dbool	Deletes the registry value

`READSTRING (Skey,Sident,Sdef) : S`	ReadString returns a string value from a specified data value associated with a key. Skey is a string that identifies the key from which to retrieve a data value. Sident is a string that identifies the name of the data value to return. Sdef is a string value to use if there is no key corresponding to Skey or no data value corresponding to Sident.
`WRITESTRING (Skey,Sident,S)`	WriteString stores a string value in a data value associated with a specified key. Skey identifies the key into which to store a data value. Sident identifies the name of the data value into which to write. S is the value to write into the data value.
`READINT (Skey,Sident,Ddef) : Dint`	ReadInt returns an integer value from a specified data value associated with a key. Skey is a string that identifies the key from which to retrieve a data value. Sident is a string that identifies the name of the data value to return. Ddef is a numeric value to use if there is no key corresponding to Skey or no data value corresponding to Sident.
`WRITEINT (Skey,Sident,Dint)`	WriteInt stores an integer value in a data value associated with a specified key. Skey identifies the key into which to store a data value. Sident identifies the name of the data value into which to write. Dint is the value to write into the data value
`READBOOL (Skey,Sident,Ddef) : Dbool`	ReadBool returns a boolean value from a specified data value associated with a key. Skey is a string that identifies the key from which to retrieve a data value. Sident is a string that identifies the name of the data value to return. Ddef is a boolean value to use if there is no key corresponding to Skey or no data value corresponding to Sident.
`WRITEBOOL (Skey,Sident,Dbool)`	WriteBool stores a boolean value in a data value associated with a specified key. Skey identifies the key into which to store a data value. Sident identifies the name of the data value into which to write. Dbool the value to write into the data value
`READDATETIME (Skey,Sident,Ddef) : Ddatetime`	ReadDateTime returns a datetime value from a specified data value associated with a key. Skey is a string that identifies the key from which to retrieve a data value. Sident is a string that identifies the name of the data value to return. Ddef is a datetime value to use if there is no key corresponding to Skey or no data value corresponding to Sident.
`WRITEDATETIME (Skey,Sident,Ddatetime)`	WriteDateTime stores a datetime value in a data value associated with a specified key. Skey identifies the key into which to store a data value. Sident identifies the name of the data value into which to write. Ddatetime is the value to write into the data value
`READNUM (Skey,Sident,Ddef) : D`	ReadNum returns a numeric value from a specified data value associated with a key. Skey is a string that identifies the key from which to retrieve a data value. Sident is a string that identifies the name of the data value to return. Ddef is a numeric value to use if there is no key corresponding to Skey or no data value corresponding to Sident.
`WRITENUM (Skey,Sident,D)`	WriteNum stores a numeric value in a data value associated with a specified key. Skey identifies the key into which to store a data

	value. Sident identifies the name of the data value into which to write. D is the value to write into the data value
`KEYEXISTS (Skey) : Dbool`	Verifies if the key Skey exists.
`VALUEEXISTS (Skey,Svalue) : Dbool`	Verifies if the value Svalue exists.
Properties	
`FILENAME`	Returns the base key (the key value passed as an argument to the constructor CREATE)

Important: Do not mess around with the Windows registry - you can corrupt your system !

2.7.3.6. The STRINGS class

The STRINGS class implements an in-memory list of strings. (eg. text)

It can be used to :

maintain a list of string values
to sort a list of string values
load textfiles into memory
create a textfile

Example:
```
myText := Strings.Create();
myText.Add('Hello');
myText.Add('from');
myText.Add('JoPPS-Script');
myText.SaveToFile('mytext.txt');
myText.Free();
```

Overview

Constructor	
`CREATE ()`	Returns an instanciated strings object
Destructor	
`FREE ()`	Frees the strings object
Methods	
`ADD (S) : Dndx`	Add a string to the end of the list
`CLEAR ()`	Clear all the strings in the list
`DELETE (Dndx)`	Delete the string at the specified index
`EXCHANGE (Dndx1,Dndx2)`	Swap the strings at the specified indexes
`INDEXOF (S) : Dndx`	Returns the index of a specific string, -1 means the strings was not found
`INSERT (Dndx,S)`	Insert the given string at the specified index
`SORT ()`	Sort the strings in the list
`LOADFROMFILE (Sfn) : B`	Load a textfile into the list, each textline takes an entry in the list
`SAVETOFILE (Sfn) : B`	Saves the strings in the list to a textfile
`CLONE : O`	Returns a duplicate of the list
`LOADDIR( ? ) : ?`	To be documented
Properties	
`COUNT`	Returns the number of strings in the list (read-only)
`STRINGS[ndx]`	To access a specific string in the list. The index specified should be in the range 0>=ndx<Count
`TEXT`	Returns the contents of the list as a single string. Lines in the string are separated by CRLF pairs.
`DELIMITER`	Specifies the delimiter used by the DelimitedText property.
`QUOTECHAR`	Specifies the quote character used by the DelimitedText property.
`COMMATEXT`	Lists the strings in the STRINGS object in system

	data format (SDF).
DELIMITEDTEXT	Represents all the strings in the STRINGS object as a single delimited string.
STRICTDELIMITER	Determines how the Delimiter property is used.
VALUES	Returns/Sets the value for a given name when using name/value pair strings

2.7.3.7. Form objects

Form objects can be used to create simple dialogs.. for more information refer to *4. Talking to the user : Form objects*.

2.8. Using OLE servers and IDispatch interfaces

J*o*PPS-Script offers the ability to instanciate and use *objects* exposed by OLE automation servers.

OLE automation servers are *programmable* software packages ranging from easy-to-use wordprocessors (eg. MSWord) to sofisticated CAD packages.

J*o*PPS-Script can interface objects exposed by OLE automation servers using an binary interface called the IDispatch interface. The IDispatch interface makes it possible to invoke object methods and to read or write its properties.

This allows us to *use* the power offered by other software packages to perform specific tasks from within our scripts.

The possibilities are endless but one of the most useful applications of this technique is exporting results from J*o*PPS to a wordprocessor or a spreadsheet.

Before we can use an object we must instanciate it using the `START` function.
`START` will load the objects server application and return a ready to use object as an I-type (IDispatch) variable.

```
word := START("word.application");
```

We should check the returned variable to make sure it holds a valid IDispatch interface. We can do this using the `IsIDispatch` function.

```
IF !IsIDispatch(word) THEN Fatal("could not start MSWORD");
```

If J*o*PPS-Script was unable to instanciate the object (the OLE automation server) `START` will return `FALSE`.

We can use the returned variable `word` as an object. We can use it methods to instanciate new objects:

```
word.Visible := TRUE;
doc := word.Documents.Add();
```

`Documents` is a property of the object `word.`

The `Documents` property returns another word object called a collection, it represents all open documents in Word.

`Add` is a method (function) of the collection object `Documents`, it adds a new document in Word and returns a object reference to it.

The variable `doc` is an I-type object representing the newly created document.

We can consult the OLE automation servers documentation to find out more about the objects it exposes, their methods and properties.

There is no way using an object without knowing how it operates: its properties, its methods and their parameters.

2.9. Using J*o*PPS as an automation client

You can use J*o*PPS v2 as automation server. This feature enables access to J*o*PPS' features from a wide range of applications and/or development environments.

JoPPS-Scripts commands can be passed to JoPPS from the caller application to perform tasks ranging from adding records in the database to building entire J*o*PPS projects from scratch.

A J*o*PPS ole automation object (jsApp) has only a few public methods, you can use the `ExecFile` and `ExeScript` methods to call J*o*PPS-Script functions or run entire scripts.

Calling J*o*PPS from a Microsoft Office VB macro..

```
...
Dim jsApp As Object
Set jsApp = CreateObject("jopps.application")
jsApp.Show
jsApp.ExecFile('c:\joppswin\jss\my own jopps script macro.jss')
Set jslApp = Nothing
...
```

List of exported ole automation object methods :

`Build : Double`	Returns the build code
`CurrentLanguage : Integer`	Returns the current language id
`DatabaseDescription : String`	Returns the description of the currently opened database
`DatabaseIsOpen : Boolean`	Returns TRUE if a database is currently open
`DatabaseName : String`	Returns the id of the database currently open
`ExecFile  (fn : String) : Integer`	Executes a JoPPS-Script macro file (jss)
`ExecScript  (source : String) : Integer`	Executes a JoPPS-Script macro
`GetParam  (param : String) : String`	Returns the value of the specified JoPPS parameter
`GetTranslation  (lanNdx,code,sub : Integer) : String`	Finds a translation in the JoPPS language system
`LoadPath : String`	Returns the path from where the JoPPS program was run
`Minimize`	Minimizes the JoPPS window
`Path_CAD : String`	Returns the CAD path
`Path_DATA : String`	Returns the DATA path
`Path_DATABASE : String`	Returns the location of the JoPPS database files
`Path_DBX : String`	Returns the DBX path (dbx and jie files)
`Path_DRW : String`	Returns the DRW path
`Path_HLP : String`	Returns the path to the JoPPS help files
`Path_JP : String`	Returns the JP path
`Path_TMP : String`	Returns the TMP path
`Path_TXT :String`	Returns the TXT path
`ProjectIsLoaded : Boolean`	Returns TRUE if a project is loaded
`ProjectName : String`	Returns the name of the current project if any
`Restore`	Restore the JoPPS window
`Revision : Integer`	Returns the revision number
`RunningMultiUser : Boolean`	Returns TRUE if JoPPS is running in multi-user mode (network)
`Show`	Displays the JoPPS window
`Station : Integer`	Returns the build station id
`Username : String`	Returns the id of the current user
`Version : Integer`	Returns the JoPPS version

A detailed description of these functions is beyond the scope of this document. They are mentioned here fore the sake of completeness.

3. Using J*o*PPS-Script in J*o*PPS

The embeded J*o*PPS-Script interpreter in J*o*PPS extends the basic J*o*PPS-Script syntax with specific functions, constants and objects to control different parts of J*o*PPS : the JoPPS-Script macro language.

This macro language is intended to automate calculations, access the J*o*PPS database and manipulate the the project data model.

Before explaining how to create macro's in J*o*PPS we will first discuss some concepts and terminology related to the J*o*PPS program itself.

Macro (or script)

A J*o*PPS macro is a "J*o*PPS-Script" script using J*o*PPS functions written to automate specific repetitive tasks in J*o*PPS. Besides stand-alone macro scripts there are two kind of macro's in J*o*PPS :

Tool macro's and report macro's.

Refer to the *3.1. Controlling JoPPS* for more information on the subject.

Single commands can be given directly from anywhere within JoPPS using the J*o*PPS instruction window. (pressing `[CTRL][SPACE]` pops up the console window)

The messagepane

The J*o*PPS messagepane is the region of the JoPPS main window where messages are displayed.
A message consists of the message text (string) and an optional errorcode.
If the errorcode for a message in the messagepane is non-zero its is assumed to be an errormessage.

You can output a message to the messagepane calling the the function `OutputMsg`.

Related functions :

`ClearMsgPane`	Clear all messages in the messagepane. If the messagepane is currently open it will remain open.
`CloseMsgPane`	Closes the messagepane window.
`MsgPaneCount`	Returns the total number of messages in the messagepane.
`MsgPaneErrCount`	Returns the number of messages in the messagepane having an errorcode set. (different from zero)
`MsgPaneGet`	Get the message string from a message in the messagepane.
`MsgPaneGetErrCode`	Get a specific errorcode from a message in the messagepane.
`MsgPaneIsOpen`	Returns `TRUE` when the messagepane window is currently open.
`OutputMsg`	Outputs a message (or errormessage) to the messagepane.

A project

Represents an open project in J*o*PPS. Open projects are kept and maintained in the J*o*PPS projectpool.
A project contains a single projectdata object which in turn holds all the project assemblies.

A simple way to add a new assembly to the current project is by using the function `AddAssembly`.
The function `AddFramePart` can be used to add extra frameparts to the current assembly.

A newly added assembly becomes automatically the new current assembly and will be shown directly into the J*o*PPS editor.

Use the projectpool to work with open projects.

The following figure illustrates the structure of a JoPPS project:

Project					
ProjectData					
	Assembly 1				
		Framepart 1			
			FrameElement1		
			FrameElement2		
			FrameElement3		
			FrameElement4		
			Segment1		
			FrameOpening1		
				VentPart1	
					VentElement1
					VentElement2
					VentElement3
					VentElement4
					VentOpening1
			FrameOpening2		
				VentPart2	
					VentElement1
					VentElement2
					VentElement3
					VentElement4
					VentOpening1

The projectpool

Open projects in J*o*PPS are maintained in the projectpool: the projectpool is a list of all open projects. Only one project in the projectpool can be the active project. The active project is the project the user is currently working on - the active project is called the current project.

The projectpool can be used to perform file operations on open projects such as
open a new project,
save a project,
close a project,
create a new project,
make a project the current project,
etc.

Since J*o*PPS v2 we can access the projectpool using the POOL object variable.
Refer to *3.5.2. Projectpool objects - working with project objects* for more information on accessing the projectpool using objects.

You can still manage the projectpool from your scripts using the older v1 functions : (J*o*PPS v1.x)

`GetActiveProjectIndex`	Returns the projectpool index of the active project
`ProjectCount`	Returns the number of open projects (eg. the size of the projectpool)
`ProjectClose`	Closes the current project
`ProjectNew`	Creates a new project
`ProjectOpen`	Opens a project
`ProjectSave`	Save the current project
`ProjectSaveAs`	Save the current project under a new name
`SetActiveProjectIndex`	Makes a project active (eg. sets the current project)

Generating results in J*o*PPS

Generating results in J*o*PPS consists of two phases :

- The calculation phase :
  Calculations are made according to the current calculation mode (See the discussion of the calculation mode below.) The result database is updated.
  Start calculations by calling the function `Calculate`. If calculations are successful and the result database is up-to-date the report generation phase will start automatically.

- The report generation phase : (e.g. the J*o*PPS report generator)
  Updates the requested (tagged) reports. Can only be invoked when the result database is up-to-date (e.g. the calculation phase completed without errors).
  If the result database is up-to-date the report generation phase can be started calling `UpdateReports`.
  Results are written to disk when the "SaveToDisk" flag is `TRUE`.

Related functions :

`ActionsEnabled`	Returns the state of the "ActionsEnabled" flag
`Calculate`	Updates the result database.
`GetCalcMode`	Returns the current calculation mode.
`GetSaveToDisk`	Returns the state of the "SaveToDisk" flag.
`GetUI`	Returns the state of the internal UI flag. (show user-interface)
`ResultsValid`	Returns `TRUE` if the result database is up-to-date.
`SetEnableActions`	Sets the state of the "ActionsEnabled" flag.
`SetBatchParams`	Sets the parameters for calculating in batch mode.
`SetCalcMode`	Sets the calculation mode.
`SetPTableParams`	Sets the parameters for calculating pricetables.
`SetUI`	Sets the state of the internal UI flag.

`SetSaveToDisk`	Sets the state of the "SaveToDisk" flag.
`TagReport`	Tags or untags report slots to be updated.
`UpdateReports`	Runs the report phase.

Report slots

Each result report occupies what is called a "slot" in J*o*PPS. Each report slot is represented by an unique number. The `SLOT_xxxxxxxx` constants can be used to refer to specific slots. For example the constant `SLOT_OFFER` represents the standard offer report. Using the `TagReport` function we can specify which slots should be updated by the report phase (e.g. `TagReport(SLOT_OFFER,TRUE)`).

Calculation mode

The calculation mode determines how the calculation phase updates the result database.
Set the appropriate calculation mode using the function `SetCalcMode`.

Possible modes are :

Calculate the current assembly only.	`CALCMODE_GROUP`
Calculate the active project only.	`CALCMODE_PROJECT`
Calculate all projects in the projectpool. The "batch dialog" is displayed when the internal UI flag is `TRUE`. (see `SetUI`)	`CALCMODE_BATCH`
Calculate pricetables for each assembly of the active project.	`CALCMODE_PTABLE`

The report generation phase cannot be invoked when the calculation mode is `CALCMODE_PTABLE`.
Pricetable information can only be requested if the option is included in your license.

Internal flags affecting calculations are :

"ActionsEnabled" flag

An internal flag used to enable or disable the execution of actions related to interfacing machining centers. (MC)
This flag is of use only for licenses having the MC option. It can be used to control the generation of machine center instructions. If there is no need to interface with a machine center this flag can be turned off (disabled) to speed up calculations. Normally this is done manually through the J*o*PPS user-interface.
To set the flag use the function `SetEnableActions`.

"SaveToDisk" flag

"SaveToDisk" is a J*o*PPS flag indicating whether or not results generated in the report phase should be written to disk.
The state of the "SaveToDisk" flag can be changed using the function `SetSaveToDisk`.

"UI" flag

If `TRUE` the internal "UI" flag tells J*o*PPS to display the :

- Batch dialog when starting calculations and the current calculation mode is `CALCMODE_BATCH`.
- Pricetable dialog when starting calculations and the current calculation mode is `CALCMODE_PTABLE`.
- Newproject dialog when calling `ProjectNew` to create a new project.

If `FALSE` none of the above dialogs is displayed.

3.1. Controlling J*o*PPS

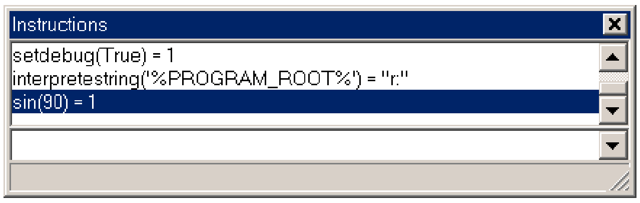
Macro's : In J*o*PPS there are three different kind of macro's you can use:

Scripts (or macro's)
JoPPS-Script routines executed in the J*o*PPS macro editor.

Tool macro's
Scripts you can add to and invoke from the J*o*PPS *Tools* menu.
Use tool macro's to automate repetitive tasks :
manage projects,
customize and run calculations,
run a set of reports and print the results,
run database maintenance tasks,
import or export price data,
perform operations on the current project or current assembly,
spawn external utility programs

Report macro's
Scripts you can associate with specific result reports.
Use report macro's to :
format and save output,
export results to other software packages.

Commands : Single JoPPS-Script instructions can be executed using the J*o*PPS Instruction Window.
(Press `[CTRL][SPACE]` to open the instruction window)



```
setdebug(True) = 1
interpretestring('%PROGRAM_ROOT%') = "r:"
sin(90) = 1
```

The instruction window can also be used as calculator.

Actions :
JoPPS-Script macro's stored in the JoPPS database.
There are three different types of actions :

Events
Actions being executed when specific predefined events occure.

Errors
Actions being triggered upon program errors.
Can be used to deal with certain errors in a non-standard way.

Decision rules
Actions implementing mainly machine related operations in order to interface machining centers. (MCs)

Output :
JoPPS-Script used in result reports

3.2. Tool macro's

To add a macro to the Tools menu :

- Pull down the "Tools" menu and select the option "Modify tools". This will popup the Modify Tools dialog.
- Press the plus button to add a tool to the list.
- Fill in the description for the new tool.
- Check the "Execute script" checkbox. This will tell J*o*PPS it is dealing with a J*o*PPS-Script.
- Enter the script code in the source editbox.
  You can load an existing script directly from file using the "Open" speedbutton.
  You can save the script pressing the "Save" speedbutton
  You can edit the script in a more advanced JScripter window pressing the "J" speedbutton.
- Add a new tool or close the dialog pressing the Ok button to confirm the new tool.

To run a macro from the Tools menu :

- Pull down the "Tools" menu.
- Select the requested tool from the dropdown menu.
- The selected tool will be executed immediately. If the script execution terminates premature due to a run-time error the JScripter window opens and the cursor is positioned at the source line causing the error.

Tool macro example :

```
IF GetActiveProjectIndex() < 0 THEN
  Fatal("No project loaded !");

OldCalcMode := GetCalcMode();
Recalc := !ResultsValid();

IF OldCalcMode <> CALCMODE_PROJECT THEN
{
  SetCalcMode(CALCMODE_PROJECT);
  Recalc := TRUE;
};

TagReport(SLOT_ALL,FALSE);
TagReport(SLOT_BILL_OF_MATERIAL,TRUE);

IF Recalc THEN Calculate() ELSE UpdateReports();

SetCalcMode(OldCalcMode);

IF HasResult(SLOT_BILL_OF_MATERIAL) THEN
{
  IF ReportInViewer() <> SLOT_BILL_OF_MATERIAL THEN
    ShowResult(SLOT_BILL_OF_MATERIAL);

  IF AskYN('Print result ?') THEN
    PrintResult(False,0);
}
ELSE
{
  Fatal("No result !");
};
```

3.3. Report macro's

Note: You need an open project in order to add or edit a report macro.

To associate a report macro with a report :

- Select the Result tab.
- Select the appropriate report in the viewer. (right-click on it)
- Click on the JS speedbutton from the report toolbar so that the J*o*PPS-Script macro dropdown menu appears. (click on the down arrow instead of on the button itself)
- Select the option "Define new macro script" from the dropdown menu.
- The JScripter window opens. Enter your script in the source pane.
- Close the JScripter window when done. J*o*PPS asks to save the changes you made. Reply yes to confirm the script source you entered.

To run a report macro :

- Press the JS button when viewing the results of a report. If a script is defined it is executed immediately. (if no script is defined the JScripter window opens enabling you to enter one..) If the script execution terminates premature due to a run-time error the JScripter window opens and the cursor is positioned at the sourceline causing the error.

Report macro example :

```
IF GetActiveProjectIndex() < 0 THEN
  Fatal("No project loaded !");

IF !ResultsValid() THEN
  Fatal("Results not up-to-date !");

slot := ReportInViewer();
IF slot < 0 THEN
  Fatal("No result in viewer !");

IF !ReportHasResult(slot) THEN
  Fatal("Report has no result !");

fn := InterpreteString("%SYSTEM_TMP%\")+ExtractFileName(InterpreteString("%REPORTDOC%"));
IF FileExists(fn) THEN DeleteFile(fn);

StrToFile(GetResultStr(slot),fn);

IF !FileExists(fn) THEN Halt; /* Oops, StrToFile failed? */

RunProgram(InterpreteString("%SYSTEM_ROOT%\notepad.exe"),fn);
```

3.4. J*o*PPS actions

Actions are JoPPS-Script macro's stored in the JoPPS action database table.

There are three different kind of actions in JoPPS :

- Events: Actions being executed when specific predefined events occure.
- Errors: Actions being triggered upon program errors. Can be used to deal with certain errors in a non-standard way.
- Decision rules: Actions implementing mainly machine related operations in order to interface machining centers. (MCs)

Normally actions are executed in background, the JoPPS cursor changes in a "J" cursor during macro execution. When an error occures the JoPPS-Script macro editor opens and positions at the line causing the error. Actions can be forced to start manually (and run visually in the JoPPS-Script macro editor) for debugging purposes. Actions can also be enabled or disabled on an per action basis.

3.4.1. Events

Actions can be used to act upon specific events within the JoPPS program, eg. a project file being opened, a new project being created, etc.

Possible events :

Predefined event actions	Parameters passed through the action object	Fired..
_AFTERAUTOSAVE	FILENAME	after the current project is saved (into an autosave file .jsv)
_AFTERCALCULATIONS	CALCMODE TERMINATIONCODE	when calculations terminate
_AFTERDATABASEOPEN	DBID	after the database is opened
_AFTERFILECLOSE	FILENAME	after a project is closed
_AFTERFILEOPEN	FILENAME PROJECT JPVERSION JPTYPE PACKCODE USERCODE SUPLID	after a project is opened
_AFTERFILESAVE	FILENAME PROJECT	after the current project is saved
_AFTERIMPORT	RID	after an import operation completes
_AFTERMODELSTORE	KIND MODELCODE MODEL	after a model (vent or frame) is stored in the library
_AFTERMODELLOAD	KIND MODELCODE	after a frame- or ventmodel is loaded from the modellibrary in the **Editor**
_AFTERNEWPROJECT	FILENAME SAVED PROJECT	after a new project is created
_AFTERRESIZETASKS		after resize frame/vent
_AFTERRUNNING	CALCMODE	when the calculations are stopped
_AFTERUPDATEPLANNING	PROJECT	after the planningtable is updated
_AFTERUPDATEREPORTS		after reports (output) are updated
_BEFOREAUTOSAVE	FILENAME PROJECT	before the current project is about to be autosaved (into an autosave file .jsv)
_BEFORECALCULATIONS	CALCMODE	before calculations start (before calculationphase)
_BEFOREDATABASECLOSE	DBID	the currently opened database is about to be closed
_BEFOREFILECLOSE	FILENAME PROJECT	before a project is closed
_BEFOREFILEOPEN	FILENAME	before a project is actually opened
_BEFOREFILESAVE	FILENAME PROJECT	before a project is saved
_BEFOREIMPORT	RID	before an import operation starts
_BEFORENEWPROJECT	FILENAME PROJECTTYPE	before a new project is created
_BEFORERESIZETASKS		before resize frame/vent
_BEFORERUNNING	CALCMODE	before starting the calculations
_BEFOREUPDATEPLANNING		before the planningtable is updated
_BEFOREUPDATEREPORTS		before reports are updated (before reportphase)
_ONAFTERACCEPT	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	after confirmation editor function
_ONAFTERCALCASSEMBLY		after calculating each assembly to be calculated
_ONAFTERCALCPROJECT		after each project to be calculated
_ONAFTERCANCEL	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	after canceling editor function
_ONAFTERREGEN	ASSEMBLY PROJECT SUCCESS	after regenerating the current assembly (editor) (SUCCESS = False when rebuild failed)
_ONAFTERESELECT	FUNID (selected	after changing selection in editor function

	function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	
_ONAFTERSENDTOPROD	BATCHID PACKID RUNTAG	after sending information to the production-follow-up module
_ONBEFOREACCEPT	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	before confirm editor function
_ONBEFORECALCASSEMBLY		before the calculation of each assembly to be calculated
_ONBEFORECALCPROJECT		before the calculation of each project to be calculated
_ONBEFORECANCEL	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	before canceling editor function
_ONBEFOREREGEN	PROJECT ASSEMBLY	before regenerating the current assembly (editor)
_ONBEFORESELECT	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	before select editor function
_ONBEFORESENDTOPROD	BATCHID PACKID RUNTAG	before sending information to the production-follow-up module
_ONCHANGED	FUNID (selected function) PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	before changing editor function takes placce
_ONEDITASSEMBLY	ACTION.ASSEMBLY (current group) ACTION.PROJECT (current project)	assembly was modified
_ONEDITPROJECT	ACTION.PROJECT (current project)	project was modified
_ONERROR	ERROR_CODE ERROR_MESSAGE ERROR_TITLE ERROR_TYPE	before default errorhandling kicks in
_ONMODELLOAD	MODEL MODELCODE	the moment frame- or ventmodel is loaded from the modellibrary
_ONMODELSTORE	MODEL MODELCODE	the moment a model (vent or frame) is stored in the library
_ONNEWASSEMBLY	ASSEMBLY	when a new assembly is created
_ONNEWFRAMEPART	FRAMEPART	when a new framepart is created
_ONPRINT	PRINT_COPIES PRINT_OPTION	before printing takes place (printing of results)
_ONPROJECTCHANGE	PROJECT	when another project becomes the current project
_ONPROJECTCHECKFAIL	PROJECT	when project checks fail (immediately after the project is opened), eg. contains errors
_ONSELECT	FUNID (selected function)	before selecting editor function takes place

	PROJECT ASSEMBLY SELECTED (# selected objects) ROWID (selected row properties list) ATOM	

Example:

Create a new action record and name it `_AFTERDATABASE` open.
Make it type 'Event' and enable the action.
Enter the following JoPPS-Script ..

```
SetWallpaperSource(HTMLToString(Action.DBID + ": I am finally open !"));
```

Post the new record and reselect the database.
Watch how the JoPPS wallpaper changes once the database reopens..

3.4.2. Trapping errors

Actions can be used to intercept error situations occuring within the J*o*PPS program.

We can run a J*o*PPS-Script macro to ignore errors or deal with errors in a non-standard way.

For example the standard errormessage error "-10027 : Project can not be read from disk"
(occuring when opening a projectfile fails) could easily be replaced by another message of our own.

For each errorcode we can provide an action record holding the macro code to be executed when the error occures. To add a macro for the -10027 error simply add a new action record and assign it the code **_10027** (underscore instead of minus!)

The next time the error occures our code will be executed first, before the standard errorhandling kicks in.

Our _10027 action is called in the same way as if it is an event. Setting the action result value in our macrocode to TRUE disables the standard J*o*PPS errorhandling.

error event	Parameters passed through the action object	Fired..
_XXXXX	ERRCODE ERRTYPE TITLE MESSAGE ATOM	When the corresponding error occures.

Example:
```
/* macro code for error -10027 */
Beep();
SetWallPaper('exploding_granate.bmp');
MsgErr2(Action.Title,Action.Message);
Action.Result := TRUE;
/* returning True means error is handled, thus disables standard errorhandling */
```

3.4.3. Decision rules

Decision rules are used to interface machining centers. (MC)

Decision rule actions are interpreted during the calculation phase when :
 the license includes the MC option,
 the internal "EnableActions" flag is TRUE
 decision rule actions are linked to the current database and/or the current project. (coded)

Decision rules are used to :
 determine the operations and positions of machining operations to be performed by machining centers.

The use and coding of decision rules is beyond the scope of this document,
refer to the *Interfacing MCs - an introduction* document for more information on this topic.

3.5. J*o*PPS related objects

J*o*PPS v2.x comes with a number of build-in object classes to :

- interface with the JoPPS database
- work with projects
- interface the modellibrary
- program simple dialogs to interact with the user
- interface machining centers

The following sections give a brief overview of these classes.

3.5.1. Database objects - interfacing the J*o*PPS database

The database object classes can be used to :

- iterate through the different databasetables of the selected JoPPS database
- find,delete,edit or insert records
- export and/or import data using J*o*PPS-Script

A database object interfaces with an underlaying table. It has an in-memory recordbuffer holding the field values for the current record.

The recordbuffer can be used to

- to find a specific record in the databasetable
- to insert a new record in the databasetable
- to edit the current record

The JIEFILE class discussed later serves a different purpose. It can be used to export data from a database object to a JIEFILE. See *3.5.1.27 The JIEFILE class* for more information.

3.5.1.1. The DBTABLE class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Methods	
`ADDFIELD`	
`ADDINDEX`	
`APPEND`	
`CANCEL`	
`CANCELRANGE`	
`CLEARFIELDS`	
`CLOSE`	
`CREATETABLE`	
`DELETE`	
`DELETETABLE`	
`EDIT`	
`EMPTYTABLE`	
`FINDKEY`	
`FINDNEAREST`	
`FIRST`	
`INSERT`	
`LAST`	
`LOCATE`	
`MOVEBY`	
`NEXT`	
`OPEN`	
`OPENEXCLUSIVE`	

POST	
PRIOR	
REFRESH	
RENAMETABLE	
SETRANGE	
Properties	
ACTIVE	
BOF	
CANMODIFY	
DATABASENAME	
EOF	
EXCLUSIVE	
EXISTS	
FIELD	
FIELDCOUNT	
FILTER	
FILTERED	
FILTEROPTIONS	
INDEXNAME	
ISEMPTY	
READONLY	
RECORDCOUNT	
STATE	
TABLENAME	Returns the physical filename of a database object
TABLEPATH	
TABLETYPE	

To be documented

3.5.1.2. The DBQUERY class

Constructor	
CREATE (?)	
Destructor	
FREE ()	
Methods	
CANCEL	
CLEARFIELDS	
CLOSE	
DELETE	
EDIT	
EXECSQL	prevents the error message ‘Error creating cursor handle’ when execute the SQL INSERT, UPDATE, DELETE, CREATE and DROP commands when using 'Open'.
FIRST	
LAST	
MOVEBY	
NEXT	
OPEN	
POST	
PRIOR	
REFRESH	
Properties	
ACTIVE	
BOF	
CANMODIFY	
DATABASENAME	

EOF	
FIELD	
FIELDCOUNT	
FILTER	
FILTERED	
FILTEROPTIONS	
ISEMPTY	
RECORDCOUNT	
REQUESTLIVE	
SQL	
STATE	

To be documented

3.5.1.3. Common methods and properties

The following properties and methods are common to all JoPPS databasetable classes.

Constructor	
`CREATE ()`	Returns an instanciated database object. Always tests for the return value of the constructor. It can be Nil !
Destructor	
`FREE ()`	Frees the database object.
Common methods	
`ASSIGN`	
`CLEAR ()`	Clears the in-memory recordbuffer.
`DELETE () : B`	Deletes the current record from the databasetable. Returns TRUE if the record was deleted successfully. The recordbuffer is updated to reflect the new current record.
`EDIT () : B`	Commit changes made to the recordbuffer to the current record. Returns TRUE if successfull.
`FIND () : B`	Finds a record. A value for the keyfields of the recordbuffer must be set before calling Find. If the record is found (result is TRUE) the recordbuffer is updated to reflect the new current record.
`FINDNEAREST ()`	Positions the current record to the record that most closely matches the key values specified in the recordbuffer. The recordbuffer is updated to reflect the new current record.
`FIRST ()`	Moves to the first record in the databasetable. The recordbuffer is updated to reflect the new current record.
`GETBOOKMARK () : Sbookmark`	Returns a bookmark for the current record. It can be used as an argument to the GotoBookmark method to return to this record at a later time. A bookmark is a string value holding the records key.
`GOTOBOOKMARK (Sbookmark) : B`	Positions the current record on the datarecord specified by Sbookmark. The recordbuffer is updated to reflect the new current record. A bookmark is a string value holding the records key.
`INSERT () : B`	Insert a new record into the databasetable. The recordbuffer is written to the new record. Returns TRUE if the record is inserted in the databasetable and the newly inserted record becomes the current record.
`LAST ()`	Moves to the last record in the databasetable. The

	recordbuffer is updated to reflect the new current record.
`LOCATE( ? ) : ?`	
`NEXT ()`	Moves to the next record in the databasetable. The recordbuffer is updated to reflect the new current record.
`PRIOR ()`	Moves to the previous record in the databasetable. The recordbuffer is updated to reflect the new current record.
`READ ()`	Rereads the current record into the recordbuffer.
`WRITE ()`	Writes the contents of the recordbuffer to the current record.
Common properties	
`BOF : B`	Is TRUE if the current record is the first record in the databasetable.
`DOPOSTCHECKS`	
`DESC [0..4]`	Record description, most databasetables have up to 5 different descriptions - one for each language in the current languageset. Some records have only one record description. (eg. contacts)
`EOF : B`	Is TRUE when the current record is the last record in the databasetable.
`RECORDCOUNT : Dcnt`	The number of records in the databasetable.
`OWNER : S`	Id of JoPPS used to made the last change to the current record.
`MODIFIED : Ddatetime`	Timestamp of last change made to the current record.
`HELPTOPIC`	
`FILTER : Dflags`	Current record filter, defines the categories set for the current record.
`REMARK : S`	Record remark.
`CODE`	
`HIDDEN`	
`READONLY`	
`RECORDCOUNT`	

To be documented

3.5.1.4. The ACCESSORIES class

Properties	
CODE	

DRAWING
    DXF_CAD
    DXF_DRW	

LINK	
KIND	
SUPPLIER	
WEIGHT	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
INFO	
TIME1	
TIME2	
COLOUR	
LEVEL	
DEPNO	
SEQNO	

FINISHES [0..99]
    FINISH
    ORDERCODE
    STOCK
    MINSIZE
    PACKSIZE
    PRICE [0..2]
        PURCHASE
        SELLING	

ACTION [0..9]
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

3.5.1.5. The ACCESSORYSETS class

Properties	
CODE	
KIND	
SENSE	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
INFO	
TIME1	
TIME2	
COLOUR	
LEVEL	
DEPNO	
SEQNO	

ACCESSORIES [0..99]
    MINW
    MAXW
    MINH
    MAXH
    CODE
        CODE
        BOOKMARK
        DEFINED
    COUNT
    NUMERATOR
    DIVISOR
    MEASURE
    COLOUR
    INFO	

ACTION [0..9]
    SYSTEM
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

3.5.1.6. The ACCESSORYTABLE class

Properties	

CODE
    SET
    WIDTH
    HEIGHT	

ERROR	
NUMERATOR	
DIVISOR	
HANDLE	
PRICEBLOCK1	
PRICEBLOCK2	
TIME1	
TIME2	

ACCESSORIES [0..99]
    SYSTEM
    CODE
        CODE
        BOOKMARK
        DEFINED
    COUNT
    INTERVAL
    COLOUR
    INFO	

ADDON [0..9]
    PROFILE
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS
    COUNT
    LINK
    NUMERATOR
    DIVISOR
    MEASURE
    ANGLE1
    ANGLE2
    COLOUR
    INFO
    LETTER	

ACTION [0..9]
    SYSTEM
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

3.5.1.7. The ACTIONS class

Properties	
CODE	
KIND	
ACTION	
EXECUTE	
MANUALRUN	
VICTIM	
REFERENCE	
NEIGHBOR	

3.5.1.8. The COMBINATIONS class

Properties	
CODE	
PROFILE1	
SYSTEM	
CODE	
BOOKMARK	
DEFINED	
WIDTH	
THICKNESS	
PROFILE2	
SYSTEM	
CODE	
BOOKMARK	
DEFINED	
WIDTH	
THICKNESS	
OVERLAP [0..2]	
ACCESSORIES [0..4]	
CODE	
CODE	
BOOKMARK	
DEFINED	
COUNT	
COLOUR	
INFO	

3.5.1.9. The CONTACTS class

Properties	
CODE	
DESC	
DRAWING DXF_CAD DXF_DRW	
CONTACT	
CONTACT2	
CONTACT3	
ADDRESS	
ZIP	
PLACE	
COUNTRY	
PHONE	
PHONE2	
PHONE3	
MOBILE	
MOBILE2	
MOBILE3	
TELEFAX	
TELEFAX2	
TELEFAX3	
EMAIL	
EMAIL2	
EMAIL3	
POBOX_ADDRESS	
POBOX_ZIP	
POBOX_PLACE	
COEF1	
COEF2	
COEF3	
PRICEGROUP	
CURRENCY	
CURRENCYPREFIX	
FACTOR	
CLIENTTYPE	
LANGUAGE	
ROUTE	
TAXNUMBER	
TAXTARIFF	
ACCOUNT	
SELLER	
ARCHITECT	
CONDITIONS	

3.5.1.10. The ENFORCEMENTS class

Properties	

CODE
    ENFORCEMENT
    ORIENTATION
    LENGTH	

ADDON [0..2]
    PROFILE
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS
    NUMERATOR
    DIVISOR
    MEASURE
    ANGLE1
    ANGLE2
    COLOUR
    INFO
    LETTER	

3.5.1.11. The FILLING class

Properties	
CODE	
DRAWING DXF_CAD DXF_DRW UDS DRW_WIDTH DRW_HEIGHT	
APPLYVARIETY	
LINK	
KIND	
SUPPLIER	
ROTATE	
DISPLAYMODE	
THICKNESS	
WIDTH	
WEIGHT	
CLEARANCE [0..19] VALUE SYSTEM	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
TIME1	
TIME2	
COLOUR	
LEVEL	
INFO	
DEPNO	
SEQNO	
SURPLUS [0..3] [0..1]	
PLACEMENT [0..9] [0..3]	
ROUNDOFF	
SURFACE	
FINISHES [0..99] FINISH ORDERCODE PRICE [0..1] PURCHASE SELLING	
NORM [0..29] MAX NEXT CODE BOOKMARK DEFINED KIND	
DIMENSION [0..2] MAX NEXT CODE BOOKMARK DEFINED KIND	
ACCESSORIES [0..4] CODE CODE BOOKMARK DEFINED COUNT COLOUR INFO	

ACTION [0..9]
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

3.5.1.12. The FINISHES class

Properties	
CODE	

DRAWING
    DXF_CAD
    DXF_DRW
    UDS
    DRW_WIDTH
    DRW_HEIGHT	

ORDERCODE	
PRICE	
FINISH0	
FINISH1	
FINISH2	
FINISH3	
RGB	

3.5.1.13. The GLAZINGBEADS class

Properties	

`CODE`
        `GLAZINGBEAD`
        `THICKNESS`	

`ACCESSORIES [0..5] [0..2] [0..1]`
        `CODE`
                `CODE`
                `BOOKMARK`
                `DEFINED`
        `COUNT`
        `MEASURE`
        `COLOUR`
        `INFO`	

`ADDON [0..5] [0..2] [0..1]`
        `PROFILE`
                `SYSTEM`
                `CODE`
                `BOOKMARK`
                `DEFINED`
                `WIDTH`
                `THICKNESS`
        `JUNCTION`
        `MEASURE`
        `BASE`
        `COUNT`
        `COLOUR`
        `INFO`
        `LETTER`	

3.5.1.14. The OPERATIONS class

Properties	
CODE	
KIND	
DRAWING DXF_CAD DXF_DRW UDS DRW_WIDTH DRW_HEIGHT	
ROUTINE	
EXECUTE	
PRICEBLOCK1	
TIME1	
PRICEBLOCK2	
TIME2	
DEPNO	
SEQNO	

3.5.1.15. The PRICEGROUPS class

Properties	
CODE GROUP BLOCK	
EXCHANGERATE	
REDUCTION	
LOSS	
PROFIT	
COLOUR	
OFFERDISCOUNT	

3.5.1.16. The PRICES class

Properties	
CODE SYSTEM MODEL	
MINW	
MAXW	
XTRW	
MINH	
MAXH	
XTRH	
MINWMINH	
MINWMINHT1	
MINWMINHT2	
MINWMINHGA	
MAXWMINH	
MAXWMINHT1	
MAXWMINHT2	
MAXWMINHGA	
XTRWMINH	
XTRWMINHT1	
XTRWMINHT2	
XTRWMINHGA	
MINWMAXH	
MINWMAXHT1	
MINWMAXHT2	
MINWMAXHGA	
MAXWMAXH	
MAXWMAXHT1	
MAXWMAXHT2	
MAXWMAXHGA	
XTRWMAXH	
XTRWMAXHT1	
XTRWMAXHT2	
XTRWMAXHGA	
MINWXTRH	
MINWXTRHT1	
MINWXTRHT2	
MINWXTRHGA	
MAXWXTRH	
MAXWXTRHT1	
MAXWXTRHT2	
MAXWXTRHGA	
XTRWXTRH	
XTRWXTRHT1	
XTRWXTRHT2	
XTRWXTRHGA	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
PRICE	
TIME1	
TIME2	
SURFACE	
GLASSAREA	
WIDTH	
HEIGHT	
INCRW	
INCRH	

ROUNDW	
ROUNDH	

3.5.1.17. The PRICESTANDARDS class

Properties	
CODE SYSTEM MODEL WIDTH HEIGHT	
GLASSAREA	
PRICEBLOCK1	
PRICEBLOCK2	
PRICE	
TIME1	
TIME2	

3.5.1.18. The PRICETARIFFS class

Properties	
CODE SYSTEM MODEL WIDTH HEIGHT	
GLASSAREA	
PRICEBLOCK1	
PRICEBLOCK2	
PRICE	
TIME1	
TIME2	

3.5.1.19. The PRODUCTS class

Properties	

CODE
        PRODUCT
        LENGTH	

DRAWING
        DXF_CAD
        DXF_DRW
        UDS
        DRW_WIDTH
        DRW_HEIGHT	

SUPPLIER	
STDLENGTH	
USELENGTH	
USABLELENGTH	
ALLOWEDLOSS	
SAWINGLOSS	
SCRAPLOSS	
LABEL	

FINISHES [0..99]
        FINISH
        ORDERCODE
        STOCK
        MINSIZE
        PACKSIZE
        PRICE [0..7]
                PRICE
                QUANTITY	

3.5.1.20. The PROFILES class

Properties	
CODE SYSTEM PROFILE	
DRAWING DXF_CAD DXF_DRW UDS DRW_WIDTH DRW_HEIGHT	
LINK	
KIND	
SUPPLIER	
PRODUCT	
WIDTH	
THICKNESS	
GEOMETRY [0..3] [0..9]	
WEIGHT	
MOMENT [0..1]	
SURFACE [0..1]	
MINUEND	
SHORTEN	
ROUND	
REBATE	
REBATE1	
REBATE2	
MARGIN	
MARGIN1	
MARGIN2	
OFFSET	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
INFO	
TIME1	
TIME2	
COLOUR	
LEVEL	
DEPNO	
SEQNO	
MINLENGTH	
INCLENGTH	
MINPRICE	
INCPRICE	
LETTER	
PROGRAMCODE	
GLAZINGBEAD	
GLAZINGBEAD1	
GLAZINGBEAD2	
ENFORCEMENT	
ENFORCEMENT1	
ENFORCEMENT2	
COMBINE SYSTEM PROFILE	
OVERMEASURE	
MINRADIUS	
PENSIZE	

MORTISELENGTH	
INTERNALLENGTH	
INTERNALMEASURE	
RESIZE	

GASKET [0..2]
        PROFILE
                SYSTEM
                CODE
                BOOKMARK
                DEFINED
                WIDTH
                THICKNESS
        MEASURE
        COLOUR
        INFO	

ADDON [0..4]
        PROFILE
                SYSTEM
                CODE
                BOOKMARK
                DEFINED
                WIDTH
                THICKNESS
        COUNT
        MEASURE
        COLOUR
        INFO
        LETTER	

JUNCTION [0..2] [0..5]
        ACCESSORIES [0..4]
                CODE
                        CODE
                        BOOKMARK
                        DEFINED
                COUNT
        MEASURE
        ACTION
                CODE
                BOOKMARK
                DEFINED
        FLAG	

ACCESSORIES [0..4]
        CODE
                CODE
                BOOKMARK
                DEFINED
        COUNT
        COLOUR
        INFO	

ACTION [0..9]
        CODE
                CODE
                BOOKMARK
                DEFINED
        REFERENCE
        FLAG
        NUMERATOR
        DIVISOR
        OFFSET	

3.5.1.21. The SYSTEMS class

Properties	
CODE	
METHOD	
GLAZINGBEADJUNCTION	
GLASSCLEARANCE	
ELASTICITYMODULUS	

FILLACCESSORY [0..19]
    ACCESSORIES[0..4]
        CODE
            CODE
            BOOKMARK
            DEFINED
        COUNT
    ACTION
        CODE
        BOOKMARK
        DEFINED
    FLAG	

FRAMEPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

FRAMEJUNCTION	

TMULLIONPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

TMULLIONJUNCTION	

VENTPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

VENTJUNCTION	

POSTPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

POSTJUNCTION	

SILLPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

ADDONPROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

INTERNALPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
DILATATIONPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
CLOSUREPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
HANDLEPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
ORIGINPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
RELATIVEPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
CROSSPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
VENTILATIONPROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	

3.5.1.22. The TASKS class

Properties	
CODE	
KIND	
BEHAVIOUR	
EXECUTE	
CHECKED	
TASKS [0..24]	
PROFILE [0..99] CODE1 SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS CODE2 SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS	
PROFILEACTION [0..9] CODE CODE BOOKMARK DEFINED WHEN	
ACCESSORIES [0..99] CODE1 CODE BOOKMARK DEFINED CODE2 CODE BOOKMARK DEFINED	
ACCESSORIESACTION [0..9] CODE CODE BOOKMARK DEFINED WHEN	
FILLING [0..99] CODE1 CODE BOOKMARK DEFINED KIND CODE2 CODE BOOKMARK DEFINED KIND	
FILLINGACTION [0..9] CODE CODE BOOKMARK DEFINED WHEN	

`WINDOWFINISH [0..99]`
`CODE1`
`CODE`
`BOOKMARK`
`DEFINED`
`CODE2`
`CODE`
`BOOKMARK`
`DEFINED`	

`WINDOWFINISHACTION [0..9]`
`CODE`
`CODE`
`BOOKMARK`
`DEFINED`
`WHEN`	

`ACTION [0..99]`
`CODE1`
`CODE`
`BOOKMARK`
`DEFINED`
`CODE2`
`CODE`
`BOOKMARK`
`DEFINED`	

`ACTIONACTION [0..9]`
`CODE`
`CODE`
`BOOKMARK`
`DEFINED`
`WHEN`	

3.5.1.23. The WINDNORM class

Properties	

CODE
        NORM
        HEIGHT	

PRESSURE	
OFFSET	

3.5.1.24. The WINDOWFINISHING class

Properties	
CODE	

DRAWING
        DXF_CAD
        DXF_DRW
        UDS
        DRW_WIDTH
        DRW_HEIGHT	

APPLYVARIETY	
LINK	
KIND	
SUPPLIER	
ROTATE	
WIDTH	
HEIGHT	
PRICEBLOCK	
PRICEBLOCK1	
PRICEBLOCK2	
TIME1	
TIME2	
COLOUR	
LEVEL	
INFO	
DEPNO	
SEQNO	
MINMEASURE	
INCMEASURE	

FINISHES [0..99]
        FINISH
        ORDERCODE
        PRICE [0..2]
                PURCHASE
                SELLING	

DIMENSION [0..2]
        MAX
        NEXT
                CODE
                BOOKMARK
                DEFINED	

ACCESSORIES [0..4]
        CODE
                CODE
                BOOKMARK
                DEFINED
        COUNT
        COLOUR
        INFO	

ADDON [0..9]
  PROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS
  COUNT
  OFFSET
  LINK
  NUMERATOR
  DIVISOR
  MEASURE
  ANGLE1
  ANGLE2
  COLOUR
  INFO
  LETTER	

ACTION [0..9]
  CODE
    CODE
    BOOKMARK
    DEFINED
  REFERENCE
  FLAG
  NUMERATOR
  DIVISOR
  OFFSET	

3.5.1.25. The FRAMES class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	

`DRAWING`
`DXF_CAD`
`DXF_DRW`
`UDS`
`XOC`
`YOC`	

`MODEL`	
`TARGETSYSTEM`	
`TASKS [0..24]`	

`ACTION [0..9]`
`CODE`
`CODE`
`BOOKMARK`
`DEFINED`
`REFERENCE`
`FLAG`
`NUMERATOR`
`DIVISOR`
`OFFSET`	

`HEIGHT`	
`WIDTH`	
`WEIGHT`	

To be documented

3.5.1.26. The VENTS class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	

`DRAWING`
`DXF_CAD`
`DXF_DRW`
`UDS`
`XOC`
`YOC`	

`MODEL`	
`ELEMENTCOUNT`	
`TARGETSYSTEM`	
`TASKS [0..24]`	

`ACTION [0..9]`
`CODE`
`CODE`
`BOOKMARK`
`DEFINED`
`REFERENCE`
`FLAG`
`NUMERATOR`
`DIVISOR`
`OFFSET`	

`HEIGHT`	
`WIDTH`	
`WEIGHT`	

To be documented

3.5.1.27. The JIEFILE class

The JIEFILE class is introduced in JoPPS version 2.71 to allow the programmer to export data from database objects to a jiefile. For the moment only exporting data is supported.

Constructor	
`CREATE (Sfn)`	Instancieert een JIEFILE object. Een bestand met als naam Sfn wordt aangemaakt. Als een bestand met dezelfde naam reeds bestaat wordt het overschreven.
Destructor	
`FREE ()`	Dealloceert het JIEFILE object. Het jie-bestand op de harde schijf wordt afgesloten.
Methods	
`Write (Odatabaseobject[,Scomment])`	Schrijft het huidige record van het doorgegeven databaseobject (Odatabaseobject) naar het JIE-bestand. Scomment is een optionele opmerking die bij het record in het JIE-bestand wordt bijgehouden.
Common properties	
`Filename : S`	De bestandsnaam van het JIE-bestand.
`RecCount : D`	Het aantal records in het JIE-bestand

Example 1: Export all contact records to a JIE-file

```
client := CONTACTS.Create();
jie := JIEFILE.Create('c:\windows\temp\mycontacts.jie');
client.First();
while !client.Eof() do
{
  jie.Write(client);
  client.Next();
};
client.Free();
jie.Free();
```

Example 2: Export the first framemodel from the framelibrary to a JIE-file

```
framelib := FRAMES.Create();
jie := JIEFILE.Create('c:\windows\temp\myframe.jie');
framelib.First();
jie.Write(framelib);
jie.Free();
framelib.Free();
```

Note: a JIE file containing a frame- or ventmodel can only contain a single record!

3.5.1.28. Database objects: An example

The following example prompts the user to enter the code for an action and add it to each filling record in the filling table.

```
new_action := AskStr('Specify action code..','','',32);
IF new_action = '' THEN halt;

f := filling.Create(); /* instanciate filling table object */

f.First(); /* position at first record */

WHILE !f.Eof DO /* iterate till at end of file */
{
  i := 0;
  WHILE i < 10 DO
  {
    IF Trim(f.action[i].Code.Code)='' THEN
      {
        f.action[i].Code.Code := new_action;
        f.Action[i].Reference := 3;
        BREAK;
      };
      i := i + 1;
  };
  f.Edit(); /* change the record */
  f.Next(); /* position at next record */
};

f.Free();
```

3.5.2. Projectpool objects - working with project objects

Hierachical overview of the projectpool and related project objects :

_PROJECTPOOL		Container class : holds all open projects. Do not instanciate but use the POOL variable to use the J*o*PPS projectpool.
PROJECT		Represents a single open project, holds information concerning the creator of the project, its changed flag, etc.
Atoms [(1)]	PROJECTDATA	Holds all technical information for a project. Holds the list of assemblies.
	ASSEMBLY	Represents a single assembly. An assembly can consist of one or more frameparts.
	FRAMEPART	Represents a framepart. A framepart can hold frameelement, segment and/or frameopening objects.
	FRAMEELEMENT	Represents a single frameelement.
	SEGMENT	Represents a segment.
	FRAMEOPENING	Represents a frameopening. Can point to a filling or a vent definition.
	VENTPART	Represents a ventpart. A ventpart can hold ventelement and/or ventopening objects.
	VENTELEMENT	Represents a single ventelement.
	VENTOPENING	Represents a ventopening. A ventopening always points to a filling. (no nested vents)

The _PROJECTPOOL class is a container holding object instances of the class project. The projectpool maintains a list of all open projects in J*o*PPS.

To use the projectpool one should never instanciate (or free) a _PROJECTPOOL object; access the projectpool via the global POOL variable.

One project in the projectpool is called the current project : the project currently selected by the user
in the J*o*PPS IDE.

The projectpool can be used to create new projects, open existing projects, edit and save existing projects and so on.

A PROJECT object holds a single PROJECTDATA class object. The PROJECTDATA class is used to hold
project related parameters and all technical data including the different assemblies.

(1) Within J*o*PPS object instances of the class PROJECTDATA, ASSEMBLY, FRAMEPART, FRAMEELEMENT, SEGMENT, FRAMEOPENING, VENTPART, VENTELEMENT or VENTOPENING are called atoms. (see *3.5.2.3. Atom objects*)

Atom objects share a number of common properties and methods.

3.5.2.1. The PROJECTPOOL class

The projectpool class is used to manage open projects in J*o*PPS.

Only one project in the projectpool can be the active project. The active project is the project the user is currently working on - this active project is called the current project.

Projectpool (holds all open projects)				
Project 1	Project 2 (=current)	Project 3	Project 4	Project 5

Do not instanciate objects of the `_PROJECTPOOL` class, the variable `POOL` can be used to access the projectpool at all times.

Methods	
`ADD ([Sfn[,Sdesc[,Sparams[,Stype[,StemplateFn]]]]]) : Dndx`	
`CLOSE ([Bdiscard]) : Dndx`	Closes the current project. The index of the new current project is returned. If Bdiscard is `TRUE` the project is <u>always</u> closed. A return value of -1 means the projectpool is empty.
`NEW ([Sfn[,Sdesc[,Sparams]]]) : Dndx`	Creates a new empty project. If specified the project is given the name Sfn and description Sdesc. Sparams is a list of initialization parameters similar to the `ProjectNew` function.
`OPEN ([Sfn]) : Dndx`	Opens the specified project (or template) and returns its index in the projectpool. (-1 if failed)
`SAVE ([bVerbose]): B`	Saves the current project. Returns `TRUE` if successful.
`SAVEAS ([Sfn[,bVerbose]]) : B`	Saves the current project under a new name. Returns `TRUE` if successful.
`POOL.SAVE ([bVerbose[,bArchive]]): B`	bArchive = True – save project and data
`POOL.SAVEAS ([Sfn[,bVerbose[,bArchive]]]) : B`	bArchive = False - save only project (default)
Properties	
`COUNT : Dcnt`	Read-only, returns the number of open projects managed in the JoPPS projectpool.
`CURRENT : Dndx`	Read-only, returns the index of the current project in the projectpool. A value of -1 means there is no current project. (and thus there are no open projects)
`CURRENTPROJECT : Oproject`	Read-only, returns the current project object.
`PROJECTS [Dndx] : Oproject`	Read-only, array holding all open project objects. Dndx ranges from `0` till `COUNT-1`.

3.5.2.2. The PROJECT class

Properties	
`CHANGED`	
`CREATED`	
`CREATOR`	
`DATABASE`	`database used to create the project`
`FILECOMMENT`	
`FILENAME`	
`FILEPATH`	`location where the project is saved`
`LASTRUN`	

`LIVE`	`True = each modification is directly adjusted in GUI (standard behaviour)`
	`False = modifications only visible in GUI after 'Live' is reactivated`

`OWNER`	`user which owns the project`
`PROJECTDATA`	
`PROJECTTYPE`	
`RESERVED`	`0 = project not locked, 1 = project locked`
`SAVED`	
`SUPPLIER`	`supplier number database`

SETUP	
SYSTEMPROFILEFINISH	
GLAZINGBEADFINISH	
ACCESSORIESFINISH	
FILLINGFINISH	
WINDOWFINISHINGFINISH	
STIFFNERFINISH	
FRAMEPROFILEFINISH	
FRAMEGLAZINGBEADFINISH	
FRAMEACCESSORIESFINISH	
FRAMEFILLINGFINISH	
FRAMEWINDOWFINISHINGFINISH	
FRAMESTIFFNERFINISH	
VENTPROFILEFINISH	
VENTGLAZINGBEADFINISH	
VENTACCESSORIESFINISH	
VENTFILLINGFINISH	
VENTWINDOWFINISHINGFINISH	
VENTSTIFFNERFINISH	
FILLING	
FILLINGINFO	
WINDOWFINISH	
WINDOWFINISHINFO	
ACCESSORIES	
ACCESSORIESINFO	
GLAZINGBEAD	
BEADTYPE	
JUNCTION	
ENFORCEMENT	
ENFORCEMENTRULE	
NORM	
SEALING	
VISUALIZATION	
RIGHTSYMBOL	
MIRRORING	
VIEWPOINT	
INTERNAL	
EXTERNAL	
FACTOR	
CURRENCYPREFIX	
CURRENCY	
LOSSTYPE	
ALLOWEDLOSS	
SAWINGLOSS	
PROFILEPRICE	
ACCESSORYPRICE	
FILLINGPRICE	
FINISHINGPRICE	
PRICEGROUP	
BLOCKLOSS	
COST1	
BLOCKCOST1	
COST2	
BLOCKCOST2	
BLOCKPLACEMENTGLAZING	
OPENORDER	
ELEMENTORDER	
PRICEFILLING[0..9]	
CODE	
CODE	
BOOKMARK	
DEFINED	
KIND	
PRICE	
PRICEBLOCK	
USER	user who last made changes
Methods	
DELETESELECTION()	Delete the selected assembly
SETCLIENT	
SELECT	Select the current assemnbly
SETRESERVED(bState,sOwner)	bState lock/unlock project (0 or 1)
	sOwner project owner (S)

3.5.2.3. Atom objects

Objects instances of the class `PROJECTDATA`, `ASSEMBLY`, `FRAMEPART`, `FRAMEELEMENT`, `SEGMENT`, `FRAMEOPENING`, `VENTPART`, `VENTELEMENT` or `VENTOPENING` are called atoms.

JoPPS uses atoms to represent windows and all their related information.

The following overview lists all properties and methods common to all atoms :

Properties	
`CHILDCOUNT`	Returns the numbers of child atoms
`CHILDREN`	Array holding all children (`0..ChildCount-1`)
`COMMENT`	Object comment

`ID`	Returns the atoms id. Each atom class has an unique id code :
	PROJECTDATA = 201
	ASSEMBLY = 202
	FRAMEPART = 203
	FRAMEELEMENT = 204
	FRAMEOPENING = 205
	SEGMENT = 206
	VENTPART = 207
	VENTELEMENT = 208
	VENTOPENIN = 209

`ISASSEMBLY`	Returns TRUE when the atom is of the ASSEMBLY class.
`ISCLOSURE`	Returns TRUE when the atom is of the VENTELEMENT class and has the function CLOSURE.
`ISDILATATION`	
`ISFICTIVE`	Returns TRUE when the atom is
`ISFRAMEELEMENT`	
`ISFRAMEOPENING`	
`ISFRAMEPART`	
`ISGENERAL`	
`ISHANDLEPROFILE`	
`ISINTERNAL`	
`ISORIGIN`	
`ISOUTERFRAME`	
`ISPROFILE`	
`ISPROJECTDATA`	Returns TRUE when the atom is of the PROJECTDATA class.
`ISRELATIVEHANDLEPROFILE`	
`ISSEGMENT`	Returns TRUE when the atom is of the SEGMENT class.
`ISTMULLION`	Returns TRUE when the atom is of the FRAMEELEMENT or VENTELEMENT class and its function is T-Mullion.
`ISVENTELEMENT`	Returns TRUE when the atom is of the VENTELEMENT class.
`ISVENTOPENING`	Returns TRUE when the atom is of the VENTOPENING class.
`ISVENTPART`	Returns TRUE when the atom is of the VENTPART class.
`ISVENTPROFILE`	Returns TRUE when the atom is of the VENTELEMENT class and its function is VENTPROFILE.
`ISLEFTSIDE`	Returns TRUE when the atom is of the FRAMEELEMENT or VENTELEMENT class and is positioned at the LEFT in the frame

ISRIGHTSIDE	Returns TRUE when the atom is of the FRAMEELEMENT or VENTELEMENT class and is positioned at the RIGHT in the frame
ISLOWERSIDE	Returns TRUE when the atom is of the FRAMEELEMENT or VENTELEMENT class and is positioned at the BOTTOM in the frame
ISUPPERSIDE	Returns TRUE when the atom is of the FRAMEELEMENT or VENTELEMENT class and is positioned at the TOP in the frame
ISHANDLESIDE	Returns TRUE when the atom is of the VENTELEMENT class and is positioned at the HANDLE side of the frame
ISHINGESIDE	Returns TRUE when the atom is of the VENTELEMENT class and is positioned at the HINGE side of the frame
PARENT	Returns the parent object
ATOMNAME	Returns the atomname for the object
Methods	
FINDATOMBYNAME	
REBUILD	

3.5.2.4. The PROJECTDATA class

Properties	
CURRENTASSEMBLY	Returns a reference to the current assembly. Returns NIL if no assemblies exist in the project (the project is empty) or the project is not the current project.
EXTRA [0..9] DESC PRICE INFO PRICEBLOCK	
PRICEBLOCK [0..99] EXCHANGERATE REDUCTION LOSS PROFIT COEF REBATE	
ADDON [0..49] COUNT PROFILE SYSTEM CODE BOOKMARK DEFINED WIDTH THICKNESS LENGTH ANGLE1 ANGLE2 FINISH COLOUR INFO PRICEBLOCK	
ACCESSORIES [0..19] COUNT CODE CODE BOOKMARK DEFINED LENGTH FINISH COLOUR INFO PRICEBLOCK	

REMAINDER [0..19] COUNT PRODUCT LENGTH FINISH COLOUR	
Methods	
ADDASSEMBLY	
CANEXPLODE()	Check whether a project / group can be split
EXPLODE('%s_%d')	Split all groups in the current project

3.5.2.5. The ASSEMBLY class

Properties	
CODE	
COUNT	
LOCKED	
MARKED	assembly selected?
POSITION	
EXTRA [0..9] DESC PRICE INFO PRICEBLOCK	
Methods	
INITIALIZE	
EXPLODE('%s_%d')	Splits a inividual group

3.5.2.6. The FRAMEPART class

Properties	
VENTCOUNT	
FRAMEWEIGHT	
GLASSWEIGHT	
TOTALWEIGHT	
MODEL SYSTEM CODE DESC BOOKMARK DEFINED	
DEFINITION X Y FROZEN WIDTH HEIGHT OFFSET LEFT RIGHT BOTTOM TOP	correcton side correcton side correcton side correcton side
LEVEL	
PRICE	
SUPPLEMENT	
TIME1	
TIME2	
INFO	
PRICEBLOCK	
BATCH	
VISUALIZATION	

ADDON [0..19]			
	PROFILE		
		SYSTEM	
		CODE	
		BOOKMARK	
		DEFINED	
		WIDTH	
		THICKNESS	
	LINK		
	LENGTH		
	OFFSET1		
	ANGLE1		
	OFFSET2		
	ANGLE2		
	FINISH		
	COLOUR		
	INFO		
	PRICEBLOCK		
SILL			
WINDOWFINISH [0..9]			
	CODE		
		CODE	
		BOOKMARK	
		DEFINED	
	WIDTH		
	HEIGHT		
	PRICE		
	FINISH		
	COLOUR		
	INFO		
	PRICEBLOCK		
ACCESSORIES [0..19]			
	COUNT		
	CODE		
		CODE	
		BOOKMARK	
		DEFINED	
	LENGTH		
	FINISH		
	COLOUR		
	INFO		
	PRICEBLOCK		
ACTION [0..9]			
	CODE		
		CODE	
		BOOKMARK	
		DEFINED	
	REFERENCE		
	FLAG		
	NUMERATOR		
	DIVISOR		
	OFFSET		
OPENORDER			
ELEMENTORDER			
WIDTH			
HEIGHT			
WIDTH[0]			
HEIGHT[0]			
WIDTH[1]			
HEIGHT[1]			
Methods			
INITIALIZE			

3.5.2.7. The FRAMEELEMENT class

Properties	
DX	
DY	
LENGTH	
SLOPE	
XB	
XE	
XM	
YB	
YE	
YM	
AUTOCORRECT	rebate length correction
CODE KIND ID	
DEFINITION FROM CODE KIND ID NUMERATOR DIVISOR MEASURE ANGLE JUNCTION CONNECTION TILL CODE KIND ID NUMERATOR DIVISOR MEASURE ANGLE JUNCTION CONNECTION GETFROM GETTILL ANGLE	
BOW	
SIZE	
COUNT	
SWAPDIRECTION	
SWAPSIDES	
BAR XB YB XE YE A B C D H XO YO R AB AE	
LINK	
LISTINDEX	

PROFILE
    SYSTEM
    CODE
    BOOKMARK
    DEFINED
    WIDTH
    THICKNESS	

FINISH	
COLOUR	
INFO	
PRICEBLOCK	

ACCESSORIES [0..1]
    CODE
        CODE
        BOOKMARK
        DEFINED
    FINISH
    COLOUR
    COUNT
    INFO
    PRICEBLOCK
    LINK	

ENFORCEMENT
    CODE
    FINISH
    COLOUR	

DIMENSION
    XB
    YB
    XE
    YE
    DX
    DY	

LABEL	
MEASURE	

ACTION [0..9]
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

Methods	
OVERLAP	
CREATEMOP	

3.5.2.8. The FRAMEOPENING class

Properties	
CODE	

DEFINITION
    FRAMECOUNT
    FRAME [0..9]
        CODE
            KIND
            ID
        PROFILE
            SYSTEM
            CODE
            BOOKMARK
            DEFINED
            WIDTH
            THICKNESS
            BAR
                XB
                YB
                XE
                YE
                A
                B
                C
                D
                H
                XO
                YO
                R
                AB
                AE
        OFFSET
        AB
        AE
    GETFRAME
    VENTCOUNT
    VENT [0..9]
        CODE
            KIND
            ID
        PROFILE
            SYSTEM
            CODE
            BOOKMARK
            DEFINED
            WIDTH
            THICKNESS
        BAR
            XB
            YB
            XE
            YE
            A
            B
            C
            D
            H
            XO
            YO
            R
            AB
            AE
        OFFSET
        AB
        AE
    GETVENT
    FROZEN
    FRAMEWIDTH
    FRAMEHEIGHT
    VENTHEIGHT
    X
    Y	

LINK	
SYSTEM	

FILLING
    CODE
    BOOKMARK
    DEFINED
    KIND	

MODEL
    CODE
    DESC
    BOOKMARK
    DEFINED	

FINISH	
COLOUR	
HEIGHT	Returns the height of the glazing
GLAZINGSIDE	
ANGLE	
OVERSIZED	
PRICE	
SUPPLEMENT	
TIME1	
TIME2	
INFO	
PRICEBLOCK	
WIDTH	Returns the width of the glazing

VENTILATION
    PROFILE
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS
    BASE
    FINISH
    COLOUR
    INFO
    PRICEBLOCK	

CROSS
    PROFILE
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS
    VERTICAL
    HORIZONTAL
    FINISH
    COLOUR
    INFO
    PRICEBLOCK	

GLAZINGBEAD
    CODE
    FINISH
    COLOUR
    BEADTYPE
    JUNCTION
    SEALING	

ACCESSORIES [0..1]
    CODE
        CODE
        BOOKMARK
        DEFINED
    FINISH
    COLOUR
    COUNT
    INFO
    PRICEBLOCK
    LINK	

ACTION [0..9]
    CODE
        CODE
        BOOKMARK
        DEFINED
    REFERENCE
    FLAG
    NUMERATOR
    DIVISOR
    OFFSET	

3.5.2.9. The SEGMENT class

Properties	

CODE
    KIND
    ID	

DEFINITION
    FROM
        CODE
            KIND
            ID
        NUMERATOR
        DIVISOR
        MEASURE
    GETFROM
    TILL
        CODE
            KIND
            ID
        NUMERATOR
        DIVISOR
        MEASURE
    GETTILL
    ANGLE	

BAR
    XB
    YB
    XE
    YE
    A
    B
    C
    D
    H
    XO
    YO
    R
    AB
    AE	

COMBINATION
    PROFILE1
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS
    PROFILE2
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS	

DIMENSION
    XB
    YB
    XE
    YE
    DX
    DY	

3.5.2.10. The VENTPART class

Properties	
CODE	
FRAMEWEIGHT	
GLASSWEIGHT	
TOTALWEIGHT	

DEFINITION
    CONTOURCOUNT
    CONTOUR [0..9]
        CODE
            KIND
            ID
        PROFILE
            SYSTEM
            CODE
            BOOKMARK
            DEFINED
            WIDTH
            THICKNESS
        BAR
            XB
            YB
            XE
            YE
            A
            B
            C
            D
            H
            XO
            YO
            R
            AB
            AE
        OFFSET
        AB
        AE
    GETCONTOUR
    CONTOURWIDTH
    CONTOURHEIGHT
    X
    Y
    FROZEN	

KIND	
LINK	
SENSE	
VISUALIZATION	
DIRECTION	
HANDLE	
HANDLESIDE	
NUMERATOR	
DIVISOR	
WIDTH	
HEIGHT	
WIDTH[0]	
HEIGHT[0]	
WIDTH[1]	
HEIGHT[1]	

OFFSET	
HANDLEACCESSORIES [0..1]	
CODE	
CODE	
BOOKMARK	
DEFINED	
COUNT	
FINISH	
COLOUR	
INFO	
PRICEBLOCK	
ACCESSORIES [0..19]	
CODE	
CODE	
BOOKMARK	
DEFINED	
COUNT	
FINISH	
COLOUR	
INFO	
PRICEBLOCK	
WINDOWFINISH [0..9]	
CODE	
CODE	
BOOKMARK	
DEFINED	
WIDTH	
HEIGHT	
PRICE	
FINISH	
COLOUR	
INFO	
PRICEBLOCK	
ACTION [0..9]	
CODE	
CODE	
BOOKMARK	
DEFINED	
REFERENCE	
FLAG	
NUMERATOR	
DIVISOR	
OFFSET	

3.5.2.11. The VENTELEMENT class

Properties	
DX	
DY	
LENGTH	
AUTOCORRECT	rebate length correction
NEIGHBOUR	
SLOPE	
XB	
XE	
XM	
YB	
YE	
YM	
CODE	
KIND	
ID	

DEFINITION
        FROM
                CODE
                        KIND
                        ID
                NUMERATOR
                DIVISOR
                MEASURE
                ANGLE
                JUNCTION
                CONNECTION
        TILL
                CODE
                        KIND
                        ID
                NUMERATOR
                DIVISOR
                MEASURE
                ANGLE
                JUNCTION
                CONNECTION
        GETFROM
        GETTILL
        ANGLE	

ANGLE	
BOW	
SIZE	
COUNT	
SWAPDIRECTION	
SWAPSIDES	

BAR
        XB
        YB
        XE
        YE
        A
        B
        C
        D
        H
        XO
        YO
        R
        AB
        AE	

LINK	

PROFILE
        SYSTEM
        CODE
        BOOKMARK
        DEFINED
        WIDTH
        THICKNESS	

FINISH	
COLOUR	
INFO	
PRICEBLOCK	

ACCESSORIES [0..1]
        CODE
                CODE
                BOOKMARK
                DEFINED
        FINISH
        COLOUR
        COUNT
        INFO
        PRICEBLOCK
        LINK	

ENFORCEMENT CODE FINISH COLOUR	
DIMENSION XB YB XE YE DX DY	
LABEL	
MEASURE	
ACTION [0..9] CODE CODE BOOKMARK DEFINED REFERENCE FLAG NUMERATOR DIVISOR OFFSET	
Methods	
OVERLAP	
CREATEMOP	

3.5.2.12. The VENTOPENING class

Properties	
`CODE`	

`DEFINITION`
    `CONTOURCOUNT`
    `CONTOUR [0..9]`
        `CODE`
            `KIND`
            `ID`
        `PROFILE`
            `SYSTEM`
            `CODE`
            `BOOKMARK`
            `DEFINED`
            `WIDTH`
            `THICKNESS`
        `BAR`
            `XB`
            `YB`
            `XE`
            `YE`
            `A`
            `B`
            `C`
            `D`
            `H`
            `XO`
            `YO`
            `R`
            `AB`
            `AE`
        `OFFSET`
        `AB`
        `AE`
    `GETCONTOUR`
    `CONTOURWIDTH`
    `CONTOURHEIGHT`
    `X`
    `Y`
    `FROZEN`	

`LINK`	

`FILLING`
    `CODE`
    `BOOKMARK`
    `DEFINED`
    `KIND`	

`FINISH`	
`COLOUR`	
`HEIGHT`	Returns the height of the glazing
`DESC`	
`GLAZINGSIDE`	
`ANGLE`	
`OVERSIZED`	
`INFO`	
`WIDTH`	Returns the width of the glazing
`PRICEBLOCK`	

`VENTILATION`
    `PROFILE`
        `SYSTEM`
        `CODE`
        `BOOKMARK`
        `DEFINED`
        `WIDTH`
        `THICKNESS`
    `BASE`
    `FINISH`
    `COLOUR`
    `INFO`
    `PRICEBLOCK`	

```
CROSS
                PROFILE
                        SYSTEM
                        CODE
                        BOOKMARK
                        DEFINED
                        WIDTH
                        THICKNESS
                VERTICAL
                HORIZONTAL
                FINISH
                COLOUR
                INFO
                PRICEBLOCK
```
```
GLAZINGBEAD
                CODE
                FINISH
                COLOUR
                BEADTYPE
                JUNCTION
                SEALING
```
```
ACCESSORIES [0..1]
                CODE
                        CODE
                        BOOKMARK
                        DEFINED
                FINISH
                COLOUR
                COUNT
                INFO
                PRICEBLOCK
                LINK
```
```
ACTION [0..9]
                CODE
                        CODE
                        BOOKMARK
                        DEFINED
                REFERENCE
                FLAG
                NUMERATOR
                DIVISOR
                OFFSET
```

3.5.2.13. Projectpool objects: An example

3.6. Running a script at login

You can let J*o*PPS start a script on start-up. This can be usefull to automatically perform administrative tasks on a regular basis.

How to specify a start-up script :

Assume you want to start a script called `WELCOME.JSS`. Copy the script file to your J*o*PPS program folder. (normally `c:\joppswin`)

Using the **-RUN** command line parameter

Add the command line parameter by right-clicking the J*o*PPS icon and selecting properties from the pop-up menu. Change the start instruction in the Target editbox so that it looks like :

```
JOPPS.EXE -RUNwelcome.jss
```

Using a command line parameter is especially useful in network configurations where each user can instate its own start-up script.

By editing the JOPPS.INI file

The JOPPS.INI file is located in the J*o*PPS program folder (eg. `c:\joppswin`).
Edit the file using a DOS textfile editor such as Windows Notepad and add the following lines :

```
[Parameters]
Run=welcome
```

If this approach is used in a network configuration all users will launch the same script when starting J*o*PPS.

The command line parameter **-RUN** overrides the JOPPS.INI file setting.

Similar to the -RUN parameter a -RUN0 could be specified. A script called by -RUN0 will be executed before login. (-RUN is called after login)

3.7. J*o*PPS-Script in HTML: the <SCRIPT> tag

You can add J*o*PPS-Script macro's to HTML report layout definition files.

J*o*PPS-Script fragments between a `<SCRIPT>` and a `</SCRIPT>` tag are executed when the resulting HTML document is loaded into the viewer.

If the opening `<SCRIPT>` tag is without `LANGUAGE` attribute the viewer assumes it has to deal with J*o*PPS-Script. However as external browsers (eg. MS Internet Explorer or Netscape) cannot interprete J*o*PPS-Script it is better to include the `LANGUAGE` attribute.

```
...
<SCRIPT LANGUAGE="JOPPS-SCRIPT">
Beep();
MsgBox("Hello from HTML");
</SCRIPT>
...
```

You can use the **-NOSCRIPTS** command line parameter to disable the execution of these embeded scripts.

Standard J*o*PPS report templates do not use the `<SCRIPT>` functionality.

3.8. Associating a script with a project template

4. Talking to the user : Form objects

4.1. The FORMSETTINGS class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`COLOUR`	
`FONT`	FACENAME COLOUR HEIGHT BOLD ITALIC
`MONITOR`	
`SCREEN`	

4.2. The FORM class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Methods	
`ADJUSTSIZE`	
`DISPLAY`	
Properties	
`CAPTION`	
`CLIENTHEIGHT`	
`CLIENTWIDTH`	
`ISDIALOG`	
`WIDTH`	
`WINHANDLE`	
`X`	
`Y`	

To be documented

4.3. The BUTTON class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`CANCEL`	
`CAPTION`	
`DEFAULT`	
`HEIGHT`	
`WIDTH`	
`X`	
`Y`	

To be documented

4.4. The CHECKBOX class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`CAPTION`	
`CHECKED`	
`HEIGHT`	
`WIDTH`	
`X`	
`Y`	

To be documented

4.5. The DIALOG class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Methods	
`ADJUSTSIZE`	
`DISPLAY`	
Properties	
`CAPTION`	
`CLIENTHEIGHT`	
`CLIENTWIDTH`	
`ISDIALOG`	
`RESULT`	
`WIDTH`	
`WINHANDLE`	
`X`	
`Y`	

To be documented

4.6. The EDITBOX class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`HEIGHT`	
`HIGH`	
`KIND`	
`LOW`	
`MAXLEN`	
`VALUE`	
`WIDTH`	
`X`	
`Y`	
`Precision`	

To be documented

4.7. The LABEL class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`CAPTION`	
`FONT`	
`PARENTCOLOR`	inherit colour from owner (yes/no)
`PARENTFONT`	inherit text properties from owner (yes/no)
`TAG`	
`HEIGHT`	
`WIDTH`	
`X`	
`Y`	

To be documented

4.8. The LISTBOX class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Methods	
`ADD`	
`ASSIGN`	
`CLEAR`	
`DELETE`	
`EXCHANGE`	
`INDEXOF`	
`INSERT`	
Properties	
`ALIGN`	
`COUNT`	
`HEIGHT`	
`ITEMINDEX`	
`ITEMS`	
`WIDTH`	
`X`	
`Y`	

To be documented

4.9. The SELECTIONBOX class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Methods	
`ADD`	
`CLEAR`	
`DELETE`	
`EXCHANGE`	
`INDEXOF`	
`ITEMINDEX`	
`INSERT`	
Properties	

To be documented

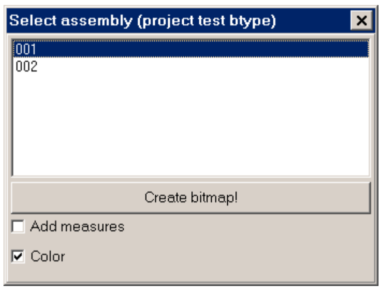
4.10. The TEXTBOX class

Constructor	
`CREATE ( ? )`	
Destructor	
`FREE ()`	
Properties	
`ALIGN`	
`HEIGHT`	
`TEXT`	
`WIDTH`	
`WORDWRAP`	
`X`	
`Y`	

To be documented

4.11. An form example

The example allows the user to choose a group from the current project and creates a bitmap of this group.



```
CurPro := GetCurrentProject();
IF CurPro = Nil THEN halt; /* no project loaded */
fn0 := InterpreteString('%PATH_OUTPUT%')+'\';
fn1 := ChangeFileExt(ExtractFilename(CurPro.Filename),'');

/* build list with assembly codes */
glst := Strings.Create();
i := 0;
WHILE i < CurPro.ProjectData.ChildCount DO
{
  grp := CurPro.ProjectData.Children[i];
  s := grp.Code;
  IF grp.Desc <> '' THEN s := s + ' : ' + grp.Desc;
  glst.Add(s);
  i := i + 1;
};

/* create assembly selection dialog */
frm :=
  FORM.Create(
    'Select assembly (project '+fn1+')');
lst :=
  LISTBOX.Create(
    frm,4,4,frm.ClientWidth - 8,frm.ClientHeight - 12 - 34 - 28 - 28,glst);
btn :=
  BUTTON.Create(
    frm,BUTTON_OK,'Create bitmap!',4,4+lst.Height+4,frm.ClientWidth - 8,30);
chkBemaat :=
  CHECKBOX.Create(
    frm,'Add measures' ,4,4+lst.Height+4+btn.Height,frm.ClientWidth - 8,24,False);
chkKleur :=
  CHECKBOX.Create(
    frm,'Colour',4,4+lst.Height+4+btn.Height+4+chkBemaat.Height,frm.ClientWidth - 8,24,True);
lst.ItemIndex := 0;

/* let user pick an assembly */
@kiesgroep:
IF frm.Display() = BUTTON_OK THEN
{
  ndx := lst.ItemIndex;
  IF ndx>=0 THEN
  {
    fn := fn0 + fn1 + '_' + CurPro.ProjectData.Children[ndx].Code + '.bmp';
    IF CreateBitmapfile (CurPro.ProjectData.Children[ndx], fn, 100, 100, chkBemaat.Checked,
          chkKleur.Checked, 1.0, 1,0, 120)
    THEN
      MsgBox('<'+fn+'> created !')
    ELSE
      MsgErr('Failed creating bitmap !');
  }
  ELSE
    MsgErr('Invalid selection !');
  GOTO kiesgroep; /* pick next assembly */
};

chkKleur.Free();
chkBemaat.Free();
btn.Free();
lst.Free();
frm.Free();
glst.Free();
```

5. Working with Machines from scripting

To be documented

6. Working with XML

Since JoPPS v 2.82 there is native support in the scripting language for manipulating, reading and writing XML files. The components presented here are part of the DOM way to traverse and query XML. Not all related XML standards are implemented here, but some are to help facilitate these kind of operations.

It needs to be noted that these components and classes are working on Strings only, and that no interpretation or immediate conversion is made. For import and export purposes, these interpretations and conversions are to be made by the writer of the script(s)!

However, conversion between strings you pass onto these functions and encoding (both read to string and write to file) of the file is automatic, including special XML encoding for special characters (i.e.: & = '&amp;').

The following chapters detail all of the XML related objects, their methods and properties. It needs to be noted that XMLNode is an object that cannot be instantiated, but that all of XMLAttribute, XMLElement, XMLCData and XMLText are all inheriting from XMLNode and are in fact all a XMLNode.

6.1. The XMLDocument class

An `XMLDocument` is a wrapper for all extra functionalities that are not related to nodes and elements. Loading and Saving of XML documents happens through this object, and the creation of new documents also.

You can also navigate your entire document from here.

6.1.1. Properties and Methods

`Create()`

This constructor creates a new `XMLDocument`, with a root element.

This means that you can call `SaveFile()` immediately after and that this call creates the following XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<root />
```

`ClearDocument();`

This method clears the entire `XMLDocument`, as if you had created a new object.

`LoadFile( fileName : string/URI): Boolean;`

This function loads a XML file into memory. The result of this method indicates if the operation was successful or not.

If the operation fails, please consult `Errors` or `HasErrors()`.

***Note!** If there is a reference to a DTD file inside the document, this DTD needs to be accessible by the parser or the result will inevitably always be false. The same can happen with badly formed or corrupt XML files or usage of another character set than specified in the encoding attribute of the XML file!*

`SaveFile( fileName : string/URI): Boolean;`

This method writes the document to disc, in the UTF-8 encoding and with readable indentation.

The result indicates if the operation was successful or not.

```
HasErrors() : Boolean;
```

This method returns true if an error occurred during the last operation on the `XMLDocument`. It would then also be possible to get an errormessage (in English) using the `Errors` property.

```
Property DocumentRoot : XMLElement (read-only)
```

This property returns the root node of the document if it exists, otherwise it returns `<Nil>`. The only way this could happen is when you read in a file that is not a valid XML file and that doesn't contain a root node. After creation, or directly after `ClearDocument()`, this returns the element for `<root />`.

```
Property Errors : String (read-only)
```

This property returns an empty string when no error occurred up until now (`XMLDocument.HasErrors() = false`). It returns a message in English with respect to the first error that occurred during the last reading or writing operation on the `XMLDocument` when `XMLDocument.HasErrors() = true` is the case.

6.2. The XMLNode class

This object type is the ancestral type for XMLAttribute, XMLElement, XMLCData and XMLText. All methods and properties as explained here are also valid for those object types. They will not be repeated there unless something is different in behaviour or result.

***Caution!** Be careful with files that were read in. There are different kind of types of nodes. Use `XMLNode.NodeType` to determine what functionality is available.*

***Caution!** The functionalities to navigate in `XMLNodes` are valid only in documents, not in `XMLNodeLists`.*

6.2.1. Constants

```
XML_ELEMENT_NODE       // = ELEMENT_NODE uit XML specificatie;
XML_ATTRIBUTE_NODE     // = ATTRIBUTE_NODE uit XML specificatie;
XML_TEXT_NODE          // = TEXT_NODE uit XML specificatie;
XML_CDATA_SECTION_NODE // = CDATA_SECTION_NODE uit XML specificatie;
```

These are constants that map directly to values that can be found in the XML DOM specification. For clarity they were fitted with a 'XML_' prefix to rule out possible double definitions.

6.2.2. Methods and properties

```
Create();
```

This method on `XMLNode` cannot ever be called from script. It cannot be created, but it can be used. A Node can represent an XML Element, a piece of text, or a comment, or an attribute, or …. `XMLNode`s are queried into a `XMLNodeList` after a XPath query for instance.

```
Property NodeName : String;
```

`NodeName` is a read/write property that returns a `String`. Depending on the derivative objecttype represented by a `XMLNode`, this property returns and sets something different.

Please consult the derivative classes for more information.

```
Property NodeValue : String;
```

`NodeName` is a read/write property that returns a `String`. Depending on the derivative objecttype represented by a `XMLNode`, this property returns and sets something different.

Please consult the derivative classes for more information.

```
Property NodeType : Integer; (read-only)
```

`NodeType` is a read-only property that returns one of the constants in the above chapter. This way you can determine what subtype of XMLNode this instance is.

```
Property ParentNode : XMLNode; (read-only)
```

A XML Document is constructed as a tree structure. Every node – Element, Comment, Attribute or something else – has a parent node to which it is child node. The big exception is the root node of the tree structure (`XMLDocument.DocumentRoot`).

This property always returns a subtype of `XMLNode` (or at least a `XMLNode`) object terug. The returned value needn't be freed later on in the script. In case of the root node, this property returns `<Nil>`.

Objects that were recently created and not attached to some other node (either explicitly or implicitly), are considered root nodes as well.

```
HasChildren() : Boolean;
```

This method returns whether this node is a so called leaf node or that it is a parent node for other nodes as well. For instance a `XMLElement` with subelements or text!

```
GetFirstChild() : XMLNode;
```

This method returns the first child node or `<Nil>` if there is none.

***Caution!** When reading files, be careful. There exist multiple types of nodes, and for now there are only some converted to a valid object (`XMLElement`, `XMLAttribute`, `XMLText` en `XMLCData`). All other node types are returned as XMLNode objects. XMLAttribute objects however are NOT fetched this way.*

```
GetLastChild() : XMLNode;
```

This method fetches the last of the dependent child nodes or `<Nil>` if there are none.

See the remark in `GetFirstChild()` for more info.

```
GetNextSibling() : XMLNode;
```

This method fetches the next childnode following the node on which you called this method in its `parentNode` or `<Nil>` if there are none.

See the remark in `GetFirstChild()` for more info.

```
GetPreviousSibling() : XMLNode;
```

This method fetches the previous childnode before the node on which you called this method in its `parentNode` or `<Nil>` if there are none.

See the remark in `GetFirstChild()` for more info.

```
GetNextInDocument() : XMLNode;
```

This method fetches the next node using a so called depth first algorithm. This means that the next `XMLNode` gives the same result as calling `GetFirstChild()` if `HasChildren()` is true. If there are no childnodes, then it returns the same as `GetNextSibling()`. If that is still `<Nil>` , it calls `ParentNode.GetNextSibling()`. It keeps on doing these steps until either `ParentNode` = `<Nil>`, in which case the result is also `<Nil>`, or a node is found.

See the remark in `GetFirstChild()` for more info.

```
GetPreviousInDocument() : XMLNode;
```

First fetches the deepest level using the last childnode of the previous sibling (if any). If no previous sibling is encountered, the parent node is returned. Returns `<Nil>` if the `XMLNode` on which this method is called is a root node.

See the remark in `GetFirstChild()` for more info.

6.3. The XMLNodeList class

`XMLNodeList`s are used throughout the DOM model to designate collections of `XMLNode` objects. Every time a method returns more than 1 `XMLNode`, this object type will be the return type.

We do not offer the possibility to create this kind of object. Note however that unless specified differently in the methods explanations, the result of such methods that return `XMLNodeList` objects, should be freed manually to prevent memory leaks!

All properties are read only!

Caution! *The functionalities to navigate in `XMLNodes` are only valid in documents, not in `XMLNodeList`s.*

6.3.1. Properties and Methods

```
Property NodesCount : Integer;
```

Returns the number of `XMLNode`s in this list. Using the indexed property `Nodes[]` will require an index in the range 0 <= index < `NodesCount`.

```
Property Nodes[ index : Integer ] : XMLNode;
```

Returns the element designated by `index`. An error will occur if `index` < 0, or `index` >= `NodesCount`. On an empty `XMLNodeList` every acces to this property will result in an error.

```
Property CurrentNode : XMLNode;
```

On creation, `CurrentNode` is made equal to the first `XMLNode ( index = 0 )`, if there is one. Calling this method can result in an error if the list is empty or one of the previous operation has put the internal pointer index to a value greater than or equal to `NodesCount` or to an `index` < 0.

See remarks on `GotoFirst()`, `GotoLast()`, `Next()` and `Previous()` to see where the internal pointer index is placed. This is an implicit call to `Nodes[]`.

`GotoFirst();`

Places the internal pointer index for `CurrentNode` to the first index. A next call to `CurrentNode` is the same as a call to `Nodes[ 0 ]`.

`GotoLast();`

Places the internal pointer index for `CurrentNode` to the last index. A next call to `CurrentNode` is the same as a call to `Nodes[ ( NodesCount - 1 ) ]`.

`IsFirst() : Boolean;`

To check if the internal pointer index is pointing to the first node (`CurrentNode` <= 0). This is always the case with newly fetched `XMLNodeList`s.

`IsLast() : Boolean;`

To check if the internal pointer index is pointing to the last node (`CurrentNode` >= `NodeCount`).

`IsValidCurrentNode() : Boolean;`

Use this method to determine if a call to `CurrentNode` would return a valid node, or a bad index error.

`IsEmpty() : Boolean;`

Returns true if the `XMLNodeList` contains no `XMLNode`s.

`Next() : XMLNode;`

First increments the internal pointer index and then returns `CurrentNode`. It can lead to errors in the script if `CurrentNode` was the last node in the nodelist before the call was made or if the list is empty.

`Previous() : XMLNode;`

First decrements the internal pointer index and then returns `CurrentNode`. It can lead to errors in the script if `CurrentNode` was the first node in the nodelist before the call was made or if the list is empty.

6.4. The XMLText class

An `XMLText` object is a derivative of `XMLNode`. This means that everything that applies to a `XMLNode`, also applies to `XMLText`

In the next piece of XML file, 'this is text' is a `XMLText` node when read from a file into an object tree.

```
<anElement>this is text</anElement>
…
```

With these kind of nodes, special XML characters are taken into account. For instance the ‘&’ character is replaced by ‘&amp;’ when the `XMLDocument` on which this text node is attached is written to a file. Leading and trailing so called ‘whitespace‘ are trimmed. If you want a piece of preformatted text inside the file, that is exactly the same when you read it out of a file next time, you need to use `XMLCData`.

6.4.1. Properties and Methods

`Create();`

You cannot create `XMLText` objects through a `XMLText.Create()` statement. This kind of objects is created automatically when you get or set the `Value` or `NodeValue` properties of an `XMLElement`, or when you call `XMLElement.AddText()` or when you read in a XML files.

However, the method is callable due to the nature of the scripting module. When you try to call this constructor, an error is generated.

`Property NodeName : String; (read-only)`

Always returns an empty string for this type of `XMLNode`.

`Property NodeValue : String;`

Returns or sets the content of the text node (in the example above, ‘this is text’). It is converted into readable text, and not in the encoding that is employed in the XML file.

`Property NodeType : Integer; (read-only)`

The value for this kind of `XMLNode` is always `XML_TEXT_NODE`.

6.5. The XMLCData class

An `XMLCData` object is a derivative of `XMLNode`. This means that everything that applies to a `XMLNode`, also applies to `XMLCData`.

In the next piece of XML file, 'this is text' is a `XMLCData` node when read from a file into an object tree.

```
…
<anElement><![CDATA[  this is text & this too

]]></anElement>
…
```

With these kind of nodes, special XML characters are taken into account. For instance the '&' character is replaced by '&amp;' when the `XMLDocument` on which this text node is attached is written to a file. Leading and trailing so called 'whitespace' are trimmed. If you want a piece of preformatted text inside the file, that is exactly the same when you read it out of a file next time, you need to use `XMLCData`.

6.5.1. Properties and Methods

```
Create();
```

You cannot create `XMLCData` objects through a `XMLCData.Create()` statement. This kind of objects is created automatically when you get or set the `Value` or `NodeValue` properties of an `XMLElement`, or when you call `XMLElement.AddCData()` or when you read in a XML files for all blocks surrounded by `<!CDATA[…]]>`.

However, the method is callable due to the nature of the scripting module. When you try to call this constructor, an error is generated.

```
Property NodeName : String; (read-only)
```

Always returns an empty string for this type of `XMLNode`.

```
Property NodeValue : String;
```

Returns or sets the content of the text node (in the example above, ' this is text & this too '+CrLf). It is converted into readable text, and not in the encoding that is employed in the XML file.

```
Property NodeType : Integer; (read-only)
```

The value for this kind of `XMLNode` is always `XML_CDATA_SECTION_NODE`.

6.6. The XMLAttribute class

An `XMLAttribute` object is a derivative of `XMLNode`. This means that everything that applies to a `XMLNode`, also applies to `XMLAttribute`.

6.6.1. Properties and Methods

```
Create();
```

Creates an attribute object that is not attached to a document or an element. Either use `XMLElement.CreateAttribute()` to create an attribute on a specific `XMLElement`, or `XMLElement.AddAttribute()` with your attribute object - that you just created with this constructor - as argument.

Do not forget to at least set the `(Node)Name` property after a call to this constructor.

```
Clear();
```

This clears the `(Node)Value` property.

```
Property NodeName : String; (read-only)
```

Returns the same value as returned by the `Name` property.

```
Property NodeValue : String;
```

Returns the same value as returned by the `Value` property.

```
Property NodeType : Integer; (read-only)
```

The value for this kind of `XMLNode` is always `XML_ATTRIBUTE_NODE`.

```
Property Name : String;
```

This property sets or gets the name of the `XMLAttribute`. E.g. in the next piece of XML file the attribute name is 'length'.

```
<profile length="7000" />
…
```

If setting the name and the new value is an empty string, an error is generated at run-time.

```
Property Value : String;
```

This property sets or gets the value of the `XMLAttribute`. E.g. in the next piece of XML file the attribute value is '7000'.

```
…
<profile length="7000" />
…
```

You can place an empty string in the value property.

6.7. The XMLElement class

An `XMLElement` object is a derivative of `XMLNode`. This means that everything that applies to a `XMLNode`, also applies to `XMLElement`.

6.7.1. Properties and Methods

```
Create();
```

Creates an element object that is not attached to a document or an element. Either use `XMLElement.CreateElement()` to create an element on a specific `XMLElement`, or `XMLElement.AddElement()` with your element object – that you just created with this constructor - as argument.

Do not forget to at least set the `(Node)Name` property after a call to this constructor.

```
Clear();
```

This method clears all underlying attributes, text or elements.

```
AddAttribute( att : XMLAttribute );
AddAttribute( att : XMLAttribute; position : Integer = -1 );
```

This method adds a previously created attribute object to the current `XMLElement.Attributes[] property on the position denoted by the position` argument. If `position` denotes an impossible position – position < 0 or position >= `AttributeCount` – then the attribute is attached as last attribute. When an attribute with this name already exists, an error occurs unless the `att` object is the same object as the one already attached. In that case, nothing happens and no error occurs..

When `att` is `<Nil>` or `att.NodeName` is empty, an error will occur.

```
AddElement( el : XMLElement );
AddElement( el : XMLElement; position : Integer = -1 );
```

This method adds a previously created element object to the current `XMLElement.Elements[] property on the position denoted by the position` argument. If `position` denotes an impossible position – position < 0 or position >= `ElementCount` – then the element is attached as last element.

When `el` is `<Nil>` or `att.NodeName` is empty, an error will occur.

```
AddCData( sText : String );
```

Adds a piece of text as the last child of an `XMLElement`. A `XMLCData` Node is attached behind the scenes with as `NodeValue` the value of `sText`.

See `XMLCData` for more information.

```
AddText( sText : String );
```

Adds a piece of text as the last child of an `XMLElement`. A `XMLText` Node is attached behind the scenes with as `NodeValue` the value of `sText`.

See `XMLText` for more information.

```
CreateAttribute( attName : string ) : XMLAttribute;
CreateAttribute( attName : string; attValue : string = '' ) : XMLAttribute;
CreateAttribute( attName : string; position : Integer = -1 ) : XMLAttribute;
CreateAttribute(attName:string; attValue:string = ''; position:Integer = -1 ) : XMLAttribute;
```

This method creates on the current `XMLElement` object an attribute with as `Name attName` and as `Value attValue`. It returns the newly created attribute after it was inserted at `position`. All other restrictions for `position` as described in `AddAttribute` are valid here as well as the same behaviour.

```
CreateElement( elName : string ) : XMLElement;
CreateElement( elName : string; elValue : string = '' ) : XMLElement;
CreateElement( elName : string; position : Integer = -1 ) : XMLElement;
CreateElement( elName : string; elValue : string = ''; position : Integer = -1 ) : XMLElement;
```

This method creates on the current `XMLElement` object an element with as `Name elName` and as `Value elValue`. It returns the newly created `XMLElement` after it was inserted at `position`. All other restrictions for `position` as described in `AddElement` are valid here as well as the same behaviour.

```
Property AttributeCount : Integer; // (read-only)
```

This property returns how many attributes are attached to this `XMLElement` object. When using the `Attributes[]` property, remember that values for `index` should always be smaller than the returned value of `AttributeCount`, but >= 0.

```
Property Attributes[ index : Integer] : XMLAttribute; // (read-only)
Property Attributes[ attName : String ] : XMLAttribute; // (read-only)
```

The way to navigate over attributes of a `XMLElement` object. The `index` argument should be between 0 (inclusive) and `AttributeCount`. When given an index out of that range, an error is generated.

When using the attName variant, this property acts the same way as `GetAttributeByName()`.

```
GetAttributeByName( attName : String ) : XMLAttribute;
```

Attempts to return an attribute with the name = `attName` on this element or `<Nil>` if none can be found. Remember that XML is case sensitive!

```
Property ElementCount : Integer; // (read-only)
```

This property returns how many elements are attached to this `XMLElement` object. When using the `Elements[]` property, remember that values for `index` should always be smaller than the returned value of `ElementCount`, but >= 0.

```
Property Elements[ index : Integer] : XMLElement; // (read-only)
```

A way to navigte over the subelements of a `XMLElement` object. The `index` argument passed to this property must lie between 0 (inclusive) and `ElementCount` or an error is generated.

```
GetElements() : XMLNodeList;
```

Returns a list with all subnodes of a `XMLElement` that are of type `XMLElement`. This list only contains `XMLNode`s of the type `XMLElement`. Remember to free the `XMLNodeList` that is returned manually!

```
GetElementsByName( elName: String ) : XMLNodeList;
```

Returns a list with all subnodes of a `XMLElement` of the type `XMLElement` and – if `elName` is a valid name for a `XMLElement` node - with `NodeName = elName`. Comparisons are – as always with XML entities - case-sensitive. Remember to free the `XMLNodeList` that is returned manually!

```
QueryForNode( xPathQuery : String ) : XMLElement
```

This method can be called on any object anywhere in a XML DOM hierarchy (on every `XMLElement` node) and is a way to enhance and speed up queries in the entire document (when `xPathQuery` starts with '/') or from that `XMLElement` on in the document.

The syntax of the `xPathQuery` argument requires a whole document of its own and is beyond the scope of this documentation. XPath is a standardized way to query XML documents in a way that SQL is for a relational database.

The result is the first (or only) `XMLNode` object that fullfills the conditions of that query or `<Nil>` if there are none.

Navigation on that returned `XMLNode` (`XMLElement`) happens as if you used the other navigational methods to that `XMLElement`. It returns the `XMLNode` object in the context of the `XMLDocument`, and not in the context of a result of a query.

If you want to use the resultset to navigate over, better use the `XMLElement.QueryForNodeList()` variant.

For more and better documentation and explanations concerning the XPath functionality, see any advanced XML, XSLT course or tutorial or use the free online course on http://www.w3schools.com/XPath/ .

This function is equivalent with the standard XML DOM function `SelectSingleNode();`.

```
QueryForNodeList( xPathQuery: String ) : XMLNodeList
```

All explanations and arguments as in `XMLElement.QueryForNode()` remain. The only differences are that this method always returns a `XMLNodeList` (that needs to be freed manually) and that the result always contains 0 or more `XMLNode` objects.

See `XMLElement.QueryForNode()` for a better explanation.

This function is equivalent with the standard XML DOM function `SelectNodes();`.

```
Property NodeName : String; (read-only)
```

Returns the same value as returned by the `Name` property.

```
Property NodeValue : String;
```

Returns the same value as returned by the `Value` property.

```
Property NodeType : Integer; (read-only)
```

The value for this kind of `XMLNode` is always `XML_ELEMENT_NODE`.

Property Name : String;

This property sets or gets the name of the `XMLElement`. E.g. in the next piece of XML file the element name is 'profile'.

```
<profile length="7000" />
…
```

When setting the name and the new value is an empty string, an error is generated at run-time.

Property Value : String;

This property sets or gets the value of the `XMLElement`. E.g. in the next piece of XML file the element value is ''.

```
…
<profile length="7000" />
…
```

Important to remember is that we currently only support so called #PCDATA. This also entails that all contents are to be translated to the encoding character set of the document. See XMLText for more information. Currently for documents that are to be created, only “UTF-8” encoding is supported. For read in XML Documents in other encodings, no guarantees are made, but chances are that they can keep this encoding on the next save of the document (even when additions have been made).

You can place an empty string in the value property or call `XMLElement.Clear()` to clear the value. This also means that when setting the value of a `XMLElement` node this is what happens:

```
…
myElement.Clear();
myElement.AddText( someValue );
…
```

6.8. Examples

XMLXPathQueryNode.jss

```
/*
This is an example of how to use the QueryForNode method.
The contents of the file we are about to read into an XMLDocument are
between the === markers below.

===
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>
<book>
  <title lang="eng">The Hichhikers' Guide To The Galaxy</title>
  <subtitle>A trilogy of 5</subtitle>
  <price>42</price>
</book>
</bookstore>
===

For more help on XPath queries, please consult and follow the tutorial on
http://www.w3schools.com/xpath/
*/

fileName := 'bookstore.xml';

myDoc := XMLDocument.Create();
if ( myDoc.LoadFile( fileName ) ) then {
  rootEl := myDoc.DocumentRoot;
  /*
     query for the one node (or the first) where you start from the root
     of the document from the root, select all title nodes that are placed
     under a book element, which has a childelement named 'price' that
     have a value greater then 35.0, and that are under a bookstore
     element on the document root
  */
  myQueryNode := rootEl.QueryForNode('/bookstore/book[price>35.0]/title');
  if (myQueryNode <> Nil ) then {
    /* this should give the following output: '"title", "Learning XML"'  */
    MsgBox('"' + myQueryNode.NodeName + '", "' + myQueryNode.Value + '"' );
  } else {
    /* mind you that the QueryForNode() method CAN return <Nil> */
    MsgBox( 'Dhow!' );
  }
  /*
    and if we wanted only the last of the nodes that fullfilled the above
    35.0 price, we would write something like this to fetch the HHGTTG
  */
  myQueryNode := rootEl.QueryForNode(
            '/bookstore/book[price>35.0][last()]/title' );
  if (myQueryNode <> Nil ) then {
    /* this should give the following output: '"title", "The Hichhikers'
       Guide To The Galaxy"'  */
    MsgBox('"' + myQueryNode.NodeName + '", "' + myQueryNode.Value + '"');
  } else {
    /* mind you that the QueryForNode() method CAN return <Nil> */
    MsgBox( 'Dhow!' );
  }

}

myDoc.Free();
```

XMLXPathQueryNodeList.jss

```
/*
This is an example of how to use the QueryForNode method.
The contents of the file we are about to read into an XMLDocument are
between the === markers below.


===
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>
<book>
  <title lang="eng">The Hichhikers' Guide To The Galaxy</title>
  <subtitle>A trilogy of 5</subtitle>
  <price>42</price>
</book>
</bookstore>
===

For more help on XPath queries, please consult and follow the tutorial on
http://www.w3schools.com/xpath/
*/

fileName := 'bookstore.xml';

myDoc := XMLDocument.Create();
if ( myDoc.LoadFile( fileName ) ) then {
  rootEl := myDoc.DocumentRoot;
  /*
     query for the one node (or the first) where you start from the root
     of the document from the root, select all title nodes that are placed
     under a book element, which has a childelement named 'price' that
     have a value greater then 35.0, and that are under a bookstore
     element on the document root
  */
  myQueryNodes := rootEl.QueryForNodeList(
         '/bookstore/book[price>35.0]/title' );

  if ( myQueryNodes.IsEmpty() = false ) then {
    myNode := myQueryNodes.CurrentNode;
    while ( true ) do {
    /* this should give the following output: '"title", "Learning XML"'  */
    /* and then : '"title", "The Hichhikers' Guide To The Galaxy"'  */
      MsgBox( '"' + myNode.Name + '", "' + myNode.Value + '"' );

      if ( myQueryNodes.IsLast() ) then break;

      myNode := myQueryNodes.Next();
    }
  } else {
    /* mind you that the QueryForNodeList() method CAN return an empty
       XMLNodeList */
    MsgBox( 'Dhow!' );
  }

}
myQueryNodes.Free(); /* an XMLNodeList needs to be freed ... and first */
myDoc.Free();
```

6.9. Error messages

Generally speaking, the messages you will deal with most of the time are those from #32701 to #32714.

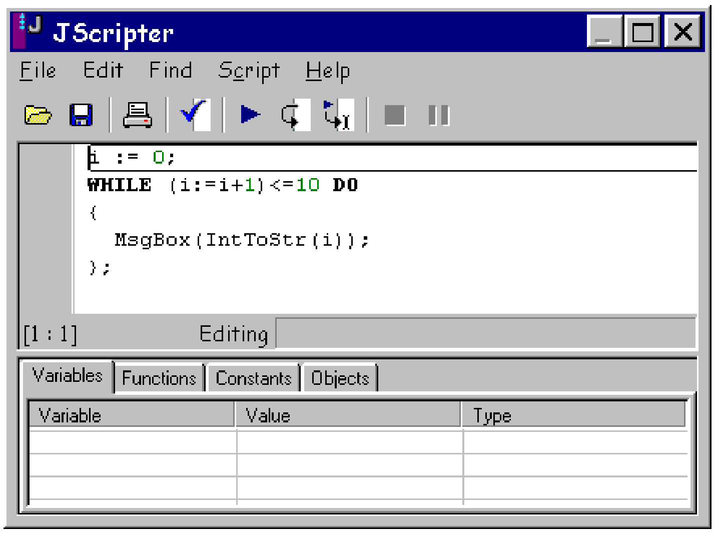
The messages from #32720 to #32731 are internally generated messages by the back-end component. These should happen seldom to never.

Message #32700 is a generic error that does not belong to one of the former 2 categories. It is something that went wrong in the scripting classes, but not in the back-end component. These would never have to happen.

Nr	Omschrijving
#32700	Generic error during processing of XML objects
#32701	You get this error when you try to construct an object using its `.Create()` method that is not allowed. Examples are `XMLNode`, `XMLNodeList`, `XMLText` en `XMLCData`.
#32702	Property not found or does not exist. See hint in errormessage.
#32703	Method not found or does not exist. See hint in errormessage.
#32704	An indexed property was given a bad value as argument. (e.g. a negative value)
#32705	An error occurred during the loading of an XML file or document.
#32706	An error occurred during the saving of an XML file or document.
#32707	If an object is `<nil>`, but at the same time of an object type, and you want to get or set a property or execute a method on it, this error message occurs.
#32708	Occurs when arguments to a method are of the wrong type. (e.g. Calling `XMLElement.AddAttribute()` with as argument `XMLElement` in stead of `XMLAttribute`.
#32709	Bad number or wrong type of arguments were input into the message call. The difference with #32708 is in the fact that all types are checked whereas in #32708 only XML objects are checked.
#32710	This property is read only and you wanted to write to it.
#32711	This property is write only and you wanted to read it.
#32712	There can be only 1 `XMLAttribute` with the same name on a `XMLElement`. When you want to add a second attribute with the same name, you get this error.
#32713	On `XMLAttribute` and `XMLElement` the `NodeName` or `Name` property cannot be empty when you add it to a `XMLDocument` or `XMLElement`. This error can occur during writing or adding of these objects to a `XMLElement`.
#32714	An argument you supplied to a method is `<Nil>`, but maybe can be of the correct type.
#32720	During reading and parsing of an `XMLDocument` this message can occur due to bad formatting or bad or unexpected characters in the file.
#32721	When you add an `XMLNode` of an incorrect type in an inappropriate place in your document.
#32722	You can only use an `XMLAttribute` on one place only in your document. If you want to add the same object on different places, you need to make another instance and copy the node.
#32723	See #32704
#32724	When you use illegal characters for a `XMLNode.NodeName` (or `Name` property) specified by the XML standard.
#32725	Some argument made something internally go wrong (in the XML objects).
#32726	Some nodetypes cannot contain data. (Zou normaliter nooit mogen voorkomen.)
#32727	When the internal processing has locked an object as read-only. Normally should never occur. The difference with #32710 is that the property you tried to use in the script is marked read/write, but internally it is not.
#32728	Queries can result in this error and designate that nothing was found due to a bad query.
#32729	The method or property called internally does not exist.
#32730	While adding data or writing to a file, both must be created (either implicitly or explicitly) for the same `XMLDocument`. Every object in the XML DOM hierarchy has a document to which it belongs. If you have 2 XML files open and you add XMLElements you found in one onto the other, this error can occur. This should normally not happen when you use the CreateXXXX methods.
#32731	When using extremely large text as value or name property for the internal processing.

7. Writing scripts using JScripter

Simple J*o*PPS-Scripts can be written and tested using the interactive J*o*PPS-Script interpreter : **JScripter**
The JScripter program has been improved to share the same look-and-feel as the JoPPS built-in macro editor of J*o*PPS v2.
In its new v2.70 incarnation the JScripter program looks like this :

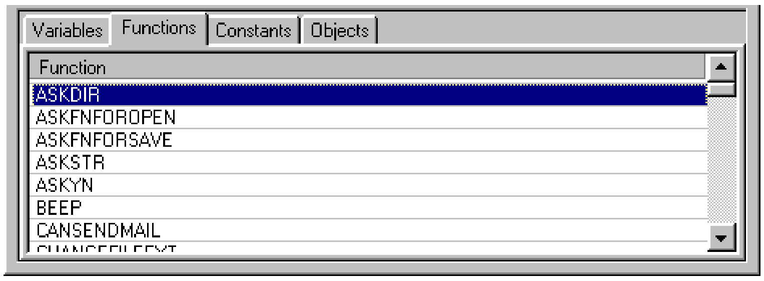

JScripter
File Edit Find Script Help
[1 : 1] Editing
Variables Functions Constants Objects
Variable Value Type

```
i := 0;
WHILE (i:=i+1)<=10 DO
{
  MsgBox(IntToStr(i));
};
```

<u>New features</u>

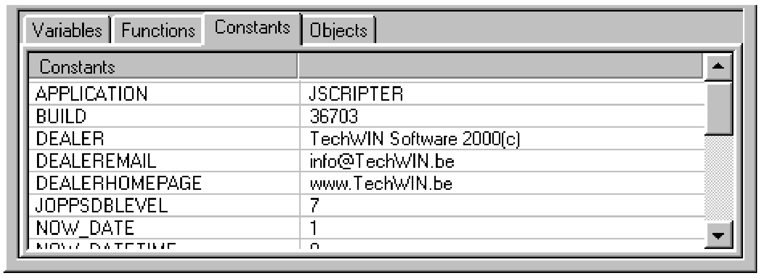
- Syntax highlighting and printing
- Improved editor with full search and replace
- Syntax check function and trace option
- Possibility to pause execution at any time
- Function, Constant and Object browser

<u>The debugwindows</u>
The debugwindows have been enhanced with tabs to give a function overview, an overview of all known constants and an object browser.



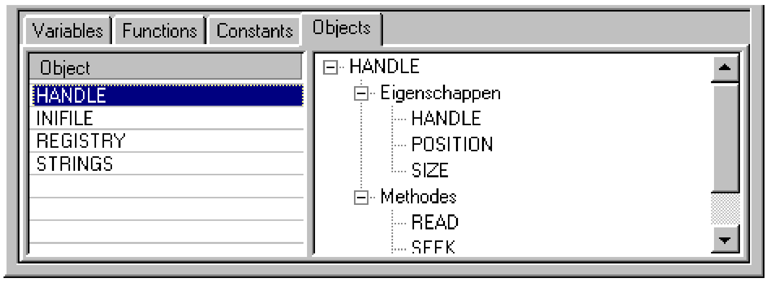
Function
ASKDIR
ASKFNFOROPEN
ASKFNFORSAVE
ASKSTR
ASKYN
BEEP
CANSENDMAIL

>the function tab



Constants	
APPLICATION	JSCRIPTER
BUILD	36703
DEALER	TechWIN Software 2000(c)
DEALEREMAIL	info@TechWIN.be
DEALERHOMEPAGE	www.TechWIN.be
JOPPSDBLEVEL	7
NOW_DATE	1

>the constants tab



>the object tab

Limitations
The standalone JScripter program does not support JoPPS specific functions or object classes :
there are no JoPPS database or form object classes available in JScripter.

Commandline support
Usage: JSCRIPTER [jssfn [-RUN]]

Specifying a filename on the commandline will load the file. The **-RUN** switch can make JScripter run the script immediatly. JScripter will start minimized and on correct termination close automatically.

8. Running J*o*PPS-Scripts from then command line: JCALL

You can execute scripts from within batch files using the JCALL command.

Usage :

```
JCALL <jss filename> [-SILENT]
```

The optional -**SILENT** parameter disables the JCALL copyright message.

Example :

KBDTEST.BAT

```
@echo off
cls
echo KEYBOARD TEST v1.0
echo.
pause
cls
jcall kbdtest.jss -silent
echo Keyboard test done.
```

KBDTEST.JSS

```
SetCaption("Testing keyboard.. press SPACE to stop");
WHILE (a := CharIn()) <> " " DO OutputMsg("Key pressed : "+a);
```

9. Function reference

This reference contains an alphabetic listing of all J*o*PPS-Script functions.

Each function description in this reference section starts with a function prototype
describing how arguments are passed.

```
FUNCTION (S, D) : I
```
platform - version spec (available since v1.2 if not specified)

The argument notation uses the following convention

D	numeric value
B	numeric value used as a boolean (1 or 0, `TRUE` or `FALSE`)
S	string value (e.g. `Sfilename` would mean a string representing a filename)
O	J*o*PPS-Script object
I	IDispatch interface

Then followed by a brief explanation what the function does, a description of it input parameters,
return value(s), etc.

Arguments between brackets [] are optional.

➘ Input parameter(s)

➘ Return value

Example :
```
word := START("word.application");
word.Visible := TRUE;
word.Documents.Add();
```

References, see also

Import remarks and pitfalls

<u>Important</u>

In the discussion of J*o*PPS related functions specific terminology is used. We assume you are familiar with
the JoPPS concepts and terminology used in the rest of this section.
If not refer to section *3. Using JoPPS-Script in JoPPS* for more information.

AA(Scode,Dcnt[,Srem[,Sparams[,Dx[,Dy[,Dwidth[,Dheight]]]]]]) : B

JoPPS – V2.82

This function is an alias for JoPPS.AddAssembly.

JoPPS.AddAssembly

AbortRun(?) : ?

JoPPS – V2.82

To be documented.

ABS(D) : Dabs

JoPPS/JCALL/JScripter

The `ABS` function returns the absolute value Dabs of the D-type argument.

➘ D-type value.

➚ Dabs, absolute value of the number D.

ACOS(D) : Dacos

JoPPS/JCALL/JScripter

The `ACOS` function calculates the inverse cosine of the given number D.

➘ D-type value must be between -1 and 1.

➚ Dacos, the inverse cosine of D in degrees and in the range [0..180].

🕮 `COS, ASIN, SIN, ATAN, TAN`

AcceptEditorFunction([Dfun]))

JoPPS-v3.30

Confirm changes in the the editor

➘ D-fun editor function identification (optional), if no editor function identification number is specified, the current function will be confirmed

🕮 `CancelEditorFunction([Dfun]))`

ActionsEnabled : Denabled

JoPPS - v2.0

The ActionsEnabled function returns the state of the internal "ActionsEnabled" flag.

➚ Returns the state of the internal "ActionsEnabled" flag.

🕮 `SetEnableActions`

AddAssembly(Scode,Dcnt[,Srem[,Sparams[,Dx[,Dy[,Dwidth[,Dheight]]]]]]) : B

JoPPS - V1.21

Adds a new assembly to the current project. The assembly is added at the end of the projects assemblylist.

➘ Scode is the assembly code for the assembly to be created,
➘ Dcnt is the assemblycount (must be 1),
➘ Srem is the assembly remark,
➘ Sparams is the assembly creation parameter string,
➘ Dx is the x position of the assembly,
➘ Dy is the y position of the assembly,
➘ Dwidth is the width of the assembly,

➘ Dheight is the height of the assembly.

The assembly creation parameter string is optional and constructed as follows :
"Ssystem,Smodel,Snewsystem"
It controls the selections of a model from the model library.
Ssystem and Smodel is the model specification, Snewsystem enables the model to be created in a different system. (if specified)

➚ Returns True if successful.

AddFramePart, AddWindowFinish, ProjectNew

`AddFramePart([Srem[,Sparams[,Dx[,Dy[,Dwidth[,Dheight]]]]]]) : B`

JoPPS - V1.21

Adds an extra framepart to the current assembly.

➘ Srem is the framepart comment,
➘ Sparams is the framepart creation parameter string,
➘ Dx is the x position of the framepart,
➘ Dy is the y position of the framepart,
➘ Dwidth is the width of the framepart,
➘ Dheight is the height of the framepart.

The framepart creation parameter string is optional and constructed as follows :
"Ssystem,Smodel,Snewsystem"
It controls the selections of a model from the model library.
Ssystem and Smodel is the model specification, Snewsystem enables the model to be created in a different system. (if specified)

➚ Returns True if successful.

AddAssembly, AddWindowFinish, ProjectNew

`AddPriceBlock (Dblock) : B`

JoPPS – V3.00

Add specified priceblock to current project.

➘ Dblock specifies code of priceblock to be added to the current project.

➚ Returns `TRUE` if priceblock is added, `FALSE` if not.

`AddWindowFinish(Dframendx,Sfinish[,Dw[,Dh[,Dprice[,Dinfo[,Dpriceblock]]]]]) : B`

JoPPS - V1.21

AddWindowFinish adds a window finish to the specified framepart of the current assembly.

➘ Dframendx identifies the target framepart where 0 is the first framepart, 1 the second and so on..,
➘ Sfinish is the window finish code to add,
➘ Dw specifies the width of the window finish,
➘ Dh specifies the height of the window finish,
➘ Dprice specifies the price of the window finish,
➘ Dinfo specifies the price info of the window finish,
➘ Dpriceblock specifies the priceblock of the window finish.

➚ Returns True if successful.

AddAssembly, AddFramePart, ProjectNew

`AF([Srem[,Sparams[,Dx[,Dy[,Dwidth[,Dheight]]]]]]) : B`

JoPPS – V2.82

This function is an alias for JoPPS.AddFramePart.

JoPPS.AddFramePart

`ASIN(D) : Dasin`

JoPPS/JCALL/JScripter

The `ASIN` function calculates the inverse sine of the given number D.

➘ D-type value, must be between -1 and 1.

➚ Dasin, the inverse sine of D in degrees and in the range [-90..90].

🕮 `SIN, ACOS, COS, ATAN, TAN`

`AskDir([Stitle[,Sdesc[,Sdefault]]]) : Sdir`

JoPPS/JCALL/JScripter

Pops up a dialog to select a subdirectory (folder).

➘ Stitle is the title displayed in the dialog captionbar (if not specified the name of the script is used),
➘ Sdesc is the optional text displayed inside the dialogbox,
➘ Sdefault is the default subdirectory.

➚ The subdirectory specified by the user or an empty string if the dialog was canceled.

`DirExists, MakeDir`

`AskFnForOpen( Stitle,Sdir,Sfn,Sext,Sfilter) : Sfn`

JoPPS/JCALL/JScripter

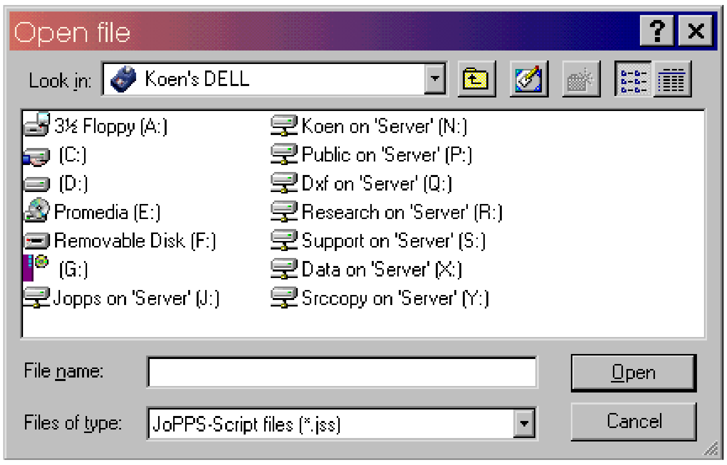
Pops up the Windows standard FileOpen dialogbox allowing the user to specify a filename.

- Stitle is the caption for the dialogbox,
- Sdir is the initial folder,
- Sfn is the default filename,
- Sext is the default file extension for the file (up to 3 characters without period),
- Sfilter is a file filter specification to fill-up the "file types" combobox.

Returns the name of the specified file or an empty string if the user canceled the input.

Example : pick a text or JS file and open it in notepad

```
filter := "JoPPS-Script files (*.jss)|*.JSS|DOS text files (*.txt)|*.TXT";
fn := AskFnForOpen("Open file","c:\joppswin\txt","","jss",filter);
IF fn <> '' THEN
{
  IF FileExists(fn) THEN
RunProgram("c:\windows\notepad.exe",fn)
  ELSE
Fatal("File <"+fn+"> not found!");
};
```



Use `FileExists` to check if the returned filename points to an existing file.

`AskFnForSave`, `FileExists`

AskFnForSave(Stitle,Sdir,Sfn,Sext,Sfilter) : Sfn
JoPPS/JCALL/JScripter

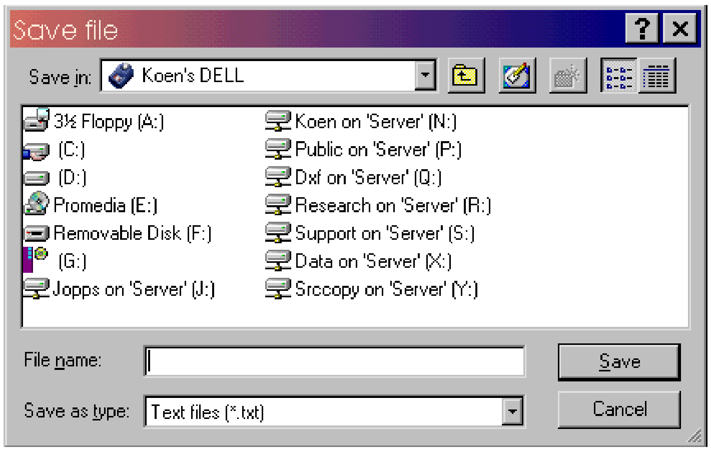
Pops up the Windows standard FileSave dialogbox allowing the user to specify a filename.

- Stitle is the caption for the dialogbox,
- Sdir is the initial folder,
- Sfn is the default filename,
- Sext is the default file extension for the file (up to 3 characters without period),
- Sfilter is a file filter specification to fill-up the "file types" combobox.

Returns the name of the specified file or an empty string if the user canceled the input.

Example : this J*o*PPS macro will save the currently shown report to a file on disk..

```
filter := "Text files (*.txt)|*.TXT|JoPPS-Script files (*.jss)|*.JSS";
fn := AskFnForSave("Save file","c:\joppswin\txt","","txt",filter);
IF fn <> '' THEN
{
  exists := FileExists(fn); /* see if file exists.. */
  IF !exists || (exists && AskYN("File <"+fn+"> already exists ! Overwrite ?")) THEN
    StrToFile(GetResultStr(),fn);
};
```



Provide an overwrite warning to prevent overwriting files.

AskFnForOpen, FileExists

AskStr([Stitle[,Sdesc[,Sdefault[,Dmaxlen[,Smask[Bpwmode[,Slow[,Shigh]]]]]]]]) : S)

JoPPS/JScripter JoPPS 3.30

Prompts the user to enter a string value.

The Stitle argument is the text displayed in the title bar of the dialog,

- Sdesc is the text displayed in the dialog,
- Sdefault is the default value for the text to enter,
- Dmaxlen is the maximum length of the returned text
- Smask is a one character string determining the characters allowed in the entered text:

"X"	any character
"!"	uppercase characters
"L"	lowercase characters
"a"	alphas only
"A"	uppercase alphas only
"l"	lowercase alphas only
"9"	digits 0..9

Specifying an empty string allows all characters to be entered (same as "X"),
If Bpwmode is TRUE the entered characters are displayed as '*' characters.

- Slow is the minimum value (empty by default)
- Shigh is the maximum value (empty by default)

Only if a range is specified for a numeric field it will be validated; at input a sound signal is given on a incorrect value, the field is colored red

- Returns the entered text.

Example :
```
password := AskStr("Login","Enter your password","",8,"",TRUE);
IF Login("ADMIN",password) <> "ADMIN" THEN
{
  Beep();
  OutputMsg("Login failed !");
  Halt;
};
```

AskYN

AskYN(SD[Breply]) : Breply

JoPPS/JCALL/JScripter

Displays a message box prompting the user to reply "Yes" (`TRUE`) or "No" (`FALSE`).

The message box title is the name of the script currently being executed.
Use `CONFIRM` instead of `ASKYN` if you want to have control over the message box title.

- One or more arguments making up the message displayed; the message displayed is the concatenation of the function arguments.
  The last argument, if multiple arguments, is a B-type parameter specifying the default action.

- Breply, `TRUE` if the user selected "Yes", `FALSE` if the user selected "No".

Example :
```
IF AskYN("Stop execution ?") THEN Halt;
IF AskYN("Update calculations for project: ",GetProjectFilename(),TRUE) THEN Calculate();
```

`Confirm`

ATAN (D) : Datan

JoPPS/JCALL/JScripter

The `ATAN` function calculates the inverse tangent of the given number D.

D-type value, must be between -1 and 1.

➚ Datan, the inverse tangent of D in degrees.

🕮 `TAN, SIN, ACOS, COS`

AtomToObj (Datom) : O

JoPPS/JCALL/JScripter - v2.0

Casts a D-type value (actually holding a pointer to an object) into an project object.
Datom must be a valid pointer to a project atom.

➚ Returns a project atom.

🕮 `ObjToInt`

This function is intended to be used in specific JoPPS result reports.

Beep ([D])

JoPPS/JCALL/JScripter

Mimics the windows MessageBeep API function.

The optional D-type argument specifies the sound type.

🕮 Refer to the Windows SDK documentation for more information. (MessageBeep)

`BrowseDataDlg(Ddlg,Scode[,…[,Dfltr[,Dlock[,Dpage[,Dhide]]]])`

JoPPS/3.27

Launches browse dialogue to select data through scripting

- Ddlg determines which data dialogue to show

  DLG_CLIENT
  DLG_FINISH
  DLG_SYSTEM
  DLG_PRODUCT
  DLG_PROFILE
  DLG_COMBINATION
  DLG_GLAZINBEAD
  DLG_REINFORCEMENT
  DLG_ACCESSORY
  DLG_SET
  DLG_ACCSET
  DLG_FILLING
  DLG_FINISHES
  DLG_PRICEBLOCK
  DLG_PRICE
  DLG_PRICESTANDARD
  DLG_PRICETARIFF
  DLG_NORM
  DLG_TASK
  DLG_ACTION
  DLG_OPERATION
  DLG_UPROFILE
  DLG_UFILLING
  DLG_CEDATA
  DLG_JOB
  DLG_FRAME
  DLG_VENT

- SCode,… key field values to show a specific record
  The number and the type of fields will depend on the key fields of the table. If the key field values is blank / zero the first record is selected
- Dfltr determines which categories will be activated
  32-bit value where each bit with value = 1 represents an activated category
- Dlock determines whether the user may change categories

  0 = user can modify categories
  1 = user can NOT modify categories

- Dpage determines the tab of the browser dialog to show

  PAGE_FORM
  PAGE_LIST
  PAGE_REMARK
  PAGE_OUTLINE
  PAGE_THUMBS
  PAGE_FILTERS
  PAGE_SPECIAL

➘ Dhide

True=categories will be hidden
False= categories will be visible

Example :
/* show client dialog */
sCode := 'TECHWIN';
sKey := BrowseDataDlg(DLG_CLIENT,sCode,0,True,PAGE_LIST,True);
if sKey <> '' then { ShowMessage(sKey); };

CancelEditorFunction([Dfun]))

JoPPS-v3.30

Cancel changes in the the editor

➚ D-fun editor function identification (optional), if no editor function identification number is specified, the current function will be canceled

AcceptEditorFunction([Dfun]))

Calculate ([Dmode]) : D

JoPPS

Start calculations according to the current calculation mode. Updates the result database and if successful the selected reports are regenerated.

➚ The optional Dmode argument overrides the current calculation mode. Possible values are:

Calculation mode constants	Meaning
`CALCMODE_GROUP`	calculate current assembly only
`CALCMODE_PROJECT`	calculate current project
`CALCMODE_BATCH`	calculate all open projects in batch, opens the "batch dialog"
`CALCMODE_PTABLE`	calculate pricetable information for current assembly

➚ The returned value is the calculation status, 0 means calculations completed without errors otherwise the last errorcode is returned.

Example :
```
IF !ResultsValid() THEN
{
  mode := GetCalcMode();
  IF mode <> CALCMODE_PROJECT THEN SetCalcMode(CALCMODE_PROJECT);
  Calculate();
  SetCalcMode(mode); /* restore previous calcmode */
  Kill("mode");
};
```

`SetCalcMode, SetUI, SetBatchParams, SetPTableParams, UpdateReports, ResultsValid`

Refer to *3. Using JoPPS-Script in JoPPS* for more information on the calculation mode.

CallPluginRoutine(?) : ?

JoPPS – V2.82

To be documented.

CanSendMail(?) : ?

JoPPS/JCALL/JScripter – V2.82

To be documented.

CatToBits(sCategories) : dFilter

JoPPS-v3.37

Work around for earlier use of the 'SetBit' function.

➘ *sCategories list of categories separated by commas*

➚ *dFilter result is bitset value filter*

🕮 SetBit

CEIL (D) : Dceil

JoPPS/JCALL/JScripter

Call Ceil to obtain the lowest integer value greater than or equal to D.

➘ D-typ²e value to be rounded.

➚ Dceil is the rounded value.

Example :

```
CEIL(-2.8)              returns -2
CEIL(2.8)               returns 3
CEIL(-1.0)              returns -1
```

FLOOR

ChangeFileExt (Sfn,Sext) : S

JoPPS/JCALL/JScripter

Changes the file extension for a given filename.

➘ Sfn is the given filename, Sext is the new file extension.

➚ The resulting filename.

Example :

```
/* get current result */
slot := ReportInViewer();
IF slot < 0 THEN Fatal("No result");
IF !ReportHasResult(slot) THEN Fatal("Report has no result");
/* get default fn for result */
fn := GetParam("REPORTDOC");
/* make bak file if already exists */
IF FileExists(fn) THEN
{
  bak := ChangeFileExt(fn,".bak");
  IF FileExists(bak) THEN DeleteFile(bak);
  RenameFile(fn,bak);
};

/* save result */
StrToFile(GetResultStr(slot),fn);
```

🕮 ExtractFilename, ExtractFilePath, ExpandFilename

CharIn () : S

JCALL

Reads a character from the standard input stream.

The script execution is paused until a character is available in the input stream.

➚ The returned string is one character long and holds the character read.

🕮 `KeyIn, LineIn, KeyPressed`

`CharOut (Dascii)`
`CharOut (Schar)`

JCALL

Writes a character to the standard output stream.

➘ Dascii is the ASCII value for the character to output.

➘ Schar is the character to output

🕮 `LineOut, TextOut`

`CHR (Dascii) : S`

JoPPS/JCALL/JScripter

Returns the character for a specified ASCII value.

➘ Dascii is the ordinal value of the character to return. Dascii must be in the range [0..255].

➚ The returned string contains the resulting character and is one character long.

🕮 `ORD`

`ClearMsgPane()`

JoPPS

Clears the message in the J*o*PPS message pane.

`OutputMsg, OpenMsgPane, CloseMsgPane`
🕮 see also *3. Using JoPPS-Script in JoPPS*

`CloseFile(Dh)`

JoPPS/JCALL/JScripter

Closes the file specified by the filehandle Dh.

🕮 `OpenFile, OpenRead, OpenWrite`

`CloseMsgPane()`

JoPPS

Closes the J*o*PPS message pane.

`OpenMsgPane, ClearMsgPane, OutputMsg`
🕮 see also *3. Using JoPPS-Script in JoPPS*

`ColorToString( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`CompareStr (S1,S2) : D`

JoPPS/JCALL/JScripter

Compares two strings. (case-sensitive)

➘ S1 and S2 are the strings to be compared.

➚ The result of the comparison. Returns 0 if both string arguments are equal.
If S1 is greater than S2, `CompareStr` returns an integer greater than 0.
If S1 is less than S2, `CompareStr` returns an integer less than 0.
The `CompareStr` function is <u>case-sensitive</u>.

CompareText

```
CompareText (S1,S2) : D
```

JoPPS/JCALL/JScripter

Compares two strings. (non case-sensitive)

➘ S1 and S2 are the strings to be compared.

➚ The result of the comparison. Returns 0 if both string arguments are equal.
If S1 is greater than S2, `CompareText` returns an integer greater than 0.
If S1 is less than S2, `CompareText` returns an integer less than 0.
The `CompareText` function is <u>not case-sensitive</u>.

CompareStr

`Confirm (SD[D]) : D`

JoPPS/JCALL/JScripter

AskYN variant except for the first argument: the title for the message box.

For more information refer to the function AskYN.

🕮 `AskYN`

`CopyFile (Sfrom, Sto) : B`

JoPPS/JCALL/JScripter

Duplicates a file.

➘ Sfrom is the full file specification of the file to duplicate,
Sto defines the location and fullname for the new file.

➚ Returns `TRUE` when the operation was successful.

🕮 `CopyFileTo, MoveFile, RenameFile`

💣 If the file Sto already exists it is overwritten without notice.

`CopyFileTo (Sfrom, Sdestination) : B`

JoPPS/JCALL/JScripter

Copies a file to a specified destination. The new file gets the same name as the original file Sfrom.

➘ Sfrom is the full file specification of the file to copy. You can pass filenames using wildcards as well.
➘ Sdestination is the location (folder) for the new file.

➚ Returns `TRUE` when the operation was successful.

🕮 `CopyFile, MoveFile, RenameFile`

If the destination file already exists it is overwritten without notice.

`COS (Dangle) : D`

JoPPS/JCALL/JScripter

The `COS` function calculates the cosine of a given angle D (in degrees).

➘ Dangle is the angle in degrees.

➚ The cosine of the given angle.

🕮 `ACOS, SIN, ASIN, TAN, ATAN`

`CreateBiColour(Obutton) : Odialog`

JoPPS 3.27

Obutton is the button object of the form which the Bi-Colour wizard should be linked

➚ Returns the dialog object Bi_Color wizard

Example :

```
/* change form default appearance */
FORMSETTINGS.FONT.BOLD := True;
FORMSETTINGS.FONT.ITALIC := False;
FORMSETTINGS.FONT.FACENAME := 'Calibri';
FORMSETTINGS.FONT.HEIGHT := 20;
```

```
/* create form */
/* arguments: title, width, height, sizeable */
frmColor := FORM.Create('Bi-Kleur voorbeeld',300,125,True);

/* create button */
/* arguments: owner, result, caption, x position, y position, width, height */
btnColor := BUTTON.Create(frmColor,BUTTON_OK,'',10,10,frmColor.ClientWidth-20,30);
btnAbort :=
BUTTON.Create(frmColor,BUTTON_CANCEL,'Annuleren',10,btnColor.Height+20,frmColor.ClientWidth-20,30);

/* create bi-colour wizard */
dlg := CreateBiColour(btnColor);

/* show form */
ret := frmColor.Display();
while (ret != BUTTON_CANCEL) do
{
  if (ret = BUTTON_OK) then
  {
    res := NewBiColour(dlg);
    if res <> '' then
    {
      ShowMessage('Nieuwe kleur = ' + res);
    }
    else
    {
      ShowMessage('Opdracht geannuleerd!');
    };
  };

  ret := frmColor.Display();
};

/* free objects */
btnAbort.Free();
btnColor.Free();
frmColor.Free();
```

NewBiColour(Odialog)

CreateBitmapFile(Oatom,Sfn,Dw,Dh,B,Bcolour,Dscale,Dscenario,Dview,Dresol,Dframeref,Dventref) : B

JoPPS - v2.50

Creates a bitmapfile (Windows BMP file) representing the given atom object.

- Oatom is the atom object (not a PROJECTDATA object),
- Sfn is the filename of the file to create,
- Dw and Dh specify the bitmap dimensions (width & height),
- B switches measurements on or off (False=off, True=on),
- Bcolour switches colour on or off (False=b/w, True= colour),
- Dscale if different from 1 (eg. 1/50=0,02),
- Dscenario is the scenario to use (0..7),
- Dview determines the viewers position (-1=as defined, 0=inside, 1=outside),
- Dresol specifies the output resolution (specify 120 for 120dpi)
- Dframeref specifies frame origin for position measures (-1=as defines, 1=frame, 2=part)
- Dventref specifies vent origin for position measures (-1=as defines, 0=vent, 1=frame, 2=part)

➚ Returns True if the file was created successfully.

CreateMetaFile

CreateFile (Sfn) : Dh

JoPPS/JCALL/JScripter - v2.0

Creates and opens a file.

➘ Sfn is the filename of the file to create.

➚ Returns a valid filehandle Dh or a value of -1 if failed. The file is opened in `OPENMODE_READWRITE` mode.

OpenFile, OpenRead, OpenWrite, CloseFile

Close the file using the `CloseFile` function when you are done using it.

CreateMetaFile(Oatom,Sfn,Dw,Dh,B,Bcolour,Dscale,Dscenario,Dview,Dresol,Dframeref,Dventref) : B

JoPPS - v2.50

Creates an enhanced metafile (Windows EMF file) representing the given atom object.

➘ Oatom is the atom object (not a PROJECTDATA object),
➘ Sfn is the filename of the file to create,
➘ Dw and Dh specify the bitmap dimensions (width & height),
➘ B switches measurements on or off (False=off, True=on),
➘ Bcolour switches colour on or off (False=b/w, True= colour),
➘ Dscale if different from 1 (eg. 1/50=0,02),
➘ Dscenario is the scenario to use (0..7),
➘ Dview determines the viewers position (-1=as defined, 0=inside, 1=outside),
➘ Dresol specifies the output resolution (specify 120 for 120dpi)
➘ Dframeref specifies frame origin for position measures (-1=as defines, 1=frame, 2=part)
➘ Dventref specifies vent origin for position measures (-1=as defines, 0=vent, 1=frame, 2=part)

➚ Returns True if the file was created successfully.

CreateBitmapFile

CurrentEditorFunction () : D

JoPPS/JCALL/JScripter – v3.31 P1

Retrieve active editor function

➚ return value < 0 id current active editor function
➚ return value = 0 there is no active editor function

DateTimeToStr ([Ddatetime]) : S

JoPPS/JCALL/JScripter

Converts a date-time value into a string.

➘ Ddatetime is a numeric value representing a date-time value.

If the Ddatetime argument does not contain a date value, the date displays as 00/00/00.
If the Ddatetime argument does not contain a time value, the time displays as 00:00:00 AM.

➚ The date-time represented as a string.

DateToStr, TimeToStr, FormatDateTime, Now,

DateToken (dYear,dMonth,dDay[,ddBase]) : S

JoPPS 3.33 P1/JCALL/JScripter

Generates a unique project reference.

➘ dYear is the start year
➘ dMonth is the start month
➘ dDay is the start day
➘ dBase is the base value for conversion (default = 36)

➚ The token represented as a string.

Example:

```
/* Project token lezen/schrijven     */
/* Gunter Selleslagh - october 2016  */

/* Current project */
CurPro := GetCurrentProject();
if CurPro = Nill then halt;

/* Project token opvragen */
ShowMessage('Project token = ' + CurPro.Token);

/* Project token wijzigen */
CurPro.Token := DateToken(2016,1,1,36);
ShowMessage('Project token = ' + CurPro.Token);
```

DateToStr (Ddatetime) : S

JoPPS/JCALL/JScripter

Converts the date part of a given date-time value into a string.
(using the ShortDateFormat setting from the Windows regional settings)

➘ Ddatetime is the date-time value to convert.

➚ The date string.

Now, DateToStr, DateTimeToStr, FormatDateTime

Day ([Ddatetime]) : Dday

JoPPS/JCALL/JScripter

Returns the day of the month.

If no parameter is given todays date is used as input, the optional Ddatetime argument specifies
an alternative date.

➚ Dday is the day of the month.

Now, Month, Year, DayOfWeek, WeekOfYear, DayOfYear

DayOfWeek ([Ddatetime]) : Dday

JoPPS/JCALL/JScripter

Returns the day of the week.

➘ If no parameter is given todays date is used as input, the optional Ddatetime argument specifies
an alternative date.

➚ Dday is the day of the week where Monday equals 1, Tuesday equals 2, etc.

Now, Day, Month, Year, WeekOfYear, DayOfYear

DayOfYear ([Ddatetime]) : Dday

JoPPS/JCALL/JScripter

Returns the day of the year.

➘ If no parameter is given todays date is used as input, the optional Ddatetime argument specifies an alternative date.

➚ Dday is the day of the year where the first day of the year is 1.

Example :

```
IF DayOfYear() = 1 THEN OutputMsg("HAPPY NEW YEAR !!!");
```

Now, Day, Month, Year, DayOfWeek, WeekOfYear

DeinstallImportFilter(?) : ?

JoPPS – V2.82

To be documented.

DeinstallPlugin(?) : ?

JoPPS – V2.82

To be documented.

DeleteFile (Sfn) : B

JoPPS/JCALL/JScripter

DeleteFile deletes a single file from disk.

➘ Sfn is the name for the file to delete.

➚ Returns `TRUE` if the file is erased, `FALSE` if the file cannot be deleted or does not exist.

💣 Use `DeleteFiles` to delete multiple files at once.

🕮 `DeleteFiles`

DeleteFiles (Sfn[Sfn],Brecycle,BshowUI)

JoPPS/JCALL/JScripter

Deletes one or more files at once. (Using the Windows Shell API)

➘ Pass the names of the files to delete as separate arguments, you can pass filenames using wildcards as well.

If Brecycle is `TRUE` the files are deleted but can be recycled from the Windows Recycle Bin.

If BshowUI is `TRUE` the standard Windows DeleteFile confirmation dialog is shown prior to the deletion and during the operation itself the Windows DeleteFile animation is played.

➚ TRUE if successful.

💣 Only local files can be recycled from the Windows Recycle Bin !

DeleteFile

DeleteFromProd(?) : ?

JoPPS – V2.82

To be documented.

```
DirExists (Sdir) : B
```
JoPPS/JCALL/JScripter

Returns whether or not the specified directory exists.

➘ Sdir is the directory specification.

➚ Returns `TRUE` if the specified directory exists.

FileExists, FileSearch

```
DiskFree () : D
DiskFree (DdriveId) : D
DiskDree (SdriveId) : D
```
JoPPS/JCALL/JScripter

Returns the number of bytes free on the specified drive.

If no parameter is passed the free space of the current drive is returned,

➘ DdriveId is the drive number (0=default, 1=A, 2=B, etc.),
➘ SdriveId is the drive letter (''=default, 'A'=drive A:, 'B'=drive B:, etc.)

➚ The number of bytes free on the specified drive.

DiskSize

```
DiskSize () : D
DiskSize (DdriveId) : D
DiskSize (SdriveId) : D
```
JoPPS/JCALL/JScripter

Returns the size of the specified drive in bytes.

If no parameter is passed the size of the current drive is returned,

➘ DdriveId is the drive number (0=default, 1=A, 2=B, etc.),
➘ SdriveId is the drive letter (''=default, 'A'=drive A:, 'B'=drive B:, etc.)

➚ Returns the total size of the disk in bytes.

DiskFree

```
DoExplode()
```
JoPPS – V3.27

Split all groups and change project phase to ‘Production’

```
DoTask( ? ) : ?
```
JoPPS – V2.82

To be documented.

```
DxfToBitmap(Sfn[,Dw,Dh,Dpf,Dbg]) : B
```
JoPPS – v3.11

Creates a bitmapfile (Windows BMP file) from a given DXF file in the same folder.

➘ Sfn is the filename of the DXF file to convert,
➘ Dw and Dh specify the bitmap dimensions (width & height), default 100x100 pixels
➘ Dpf specifies the pixelformat (3=8bit, 5=16bit, 6=24bit, 7=32bit), default 24bit
➘ Bbg specifies the background color (), default white

➚ Returns True if the file was created successfully.

Example: IF !DxfToBitmap(‘Filename.dxf’,1000,1000,3,COLOUR_BTNFACE) THEN ...

DxfToMetaFile

`DxfToMetafile(Sfn[,Dw,Dh]) : B`

JoPPS – v3.11

Creates a metafile (Windows EMF file) from a given DXF file in the same folder.

➘ Sfn is the filename of the DXF file to convert,
➘ Dw and Dh specify the bitmap dimensions (width & height), default 100x100 pixels

➚ Returns True if the file was created successfully.

Example:

```
IF !DxfToMetafile('Filename.dxf',1000,1000) THEN ...
```

DxfToBitmap

EncodeDate(?) : ?

JoPPS – V2.82

EditBox.Create([Ofrm[,Dx[,Dy[,Dw[,Dh,[Dkind[,Slow[,Shigh[,Dlen]]]]]]]]]) : O)

JoPPS – V3.30 P1

- Ofrm
- Dx
- Dy
- Dw
- Dh
- Dkind
- Slow is the minimum value (empty by default)
- Shigh is the maximum value (empty by default)

Only if a range is specified for a numeric field it will be validated;
at input a sound signal is given on a incorrect value, the field is colored red

- Dlen

Example :

```
/* ---------------------------------------------------------------- */
/* Example EditBox.jss                                              */
/*                                                                  */
/* Example for creation of GUI objects in scripting.                */
/*                                                                  */
/* TechWIN Software BVBA 2015 (c) - Gunter Selleslagh */
/* ---------------------------------------------------------------- */

/* change form default appearance */
FORMSETTINGS.FONT.BOLD := True;
FORMSETTINGS.FONT.ITALIC := False;
FORMSETTINGS.FONT.FACENAME := 'Calibri';
FORMSETTINGS.FONT.HEIGHT := 20;

/* create form */
/* arguments: title, width, height, sizeable */
frm := FORM.Create('Form sample',300,250,True);

/* create label */
/* arguments: owner, caption, x position, y position, width, height, kind, length */
lbl := LABEL.Create(frm,'Label sample',10,10,frm.ClientWidth-10,25);

/* create editbox */
/* arguments: owner, x position, y position, width, height, kind, length */
/* kinds: EDITBOX_STRING, EDITBOX_LOWER, EDITBOX_UPPER, EDITBOX_INTEGER, EDITBOX_DOUBLE */
edt := EDITBOX.Create(frm,10,lbl.Y+lbl.Height+10,frm.ClientWidth-10,25,EDITBOX_DOUBLE,’0’,’100’);
edt.Precision := 4;
edt.Value := 50.5;

/* show form */
frm.Display();

/* free objects */
edt.Free();
lbl.Free();
frm.Free();
```

EOF (Dh) : B

JoPPS/JCALL/JScripter

Checks if the file pointer is at the end of the file.

➘ Dh is the filehandle of the file.

➚ Returns `TRUE` if the file pointer is at the end of the file, `FALSE` if it is not.

ExecuteFile(?) : ?

JoPPS – V2.82

To be documented.

EXP (D) : D

JoPPS/JCALL/JScripter

Exp returns the value of e raised to the given power, where e is the base of the natural logarithms.

📖 `LN,LOG`

ExpandFileName (Sfn) : S

JoPPS/JCALL/JScripter

ExpandFileName expands the given filename to a fully qualified filename including drive
and path specification.

Embeded '.' and '..' directory references are removed.

➘ The file specification to expand.

➚ The resulting file specification.

📖 `ExtractFilename, ExtractFilePath, ChangeFileExt`

ExplorePath(?) : ?

JoPPS – V2.82

To be documented.

ExportFile(?) : ?

JoPPS – V2.82

To be documented.

ExtendedSyntax() : B

JoPPS/JCALL/JScripter

Returns if the script is running in an interpreter with an extended syntax. (eg. J*o*PPS and not JScripter)

➚ Returns `TRUE` if the J*o*PPS-Script syntax is extended with extra functions.

ExtractFilename (Sfn) : S

JoPPS/JCALL/JScripter

ExtractFileName extracts the name and extension parts of the given filename.

➘ Sfn is a full file specification

➚ Returns the name and extension only of the given file specification.

🕮 `ExtractFilePath, ExpandFilename, ChangeFileExt`

ExtractFilePath (Sfn) : S

JoPPS/JCALL/JScripter

ExtractFilePath extracts the drive and directory parts of the given filename.

➘ Sfn is the source file specification.

➚ The drive and directory portion of the filename.

🕮 `ExtractFilename, ExpandFilename, ChangeFileExt`

Fatal ([S])

JoPPS/JCALL/JScripter

Aborts the execution of the script and raises an runtime error #51.

➘ S is an optional user-defined errormessage.

Example:

```
word := START("word.application");
IF !IsIDispatch(word) THEN Fatal("WORD did not start !");
```

🕮 For more information refer to 2.6. Stopping script execution.

FileAge (Sfn) : Ddatetime

JoPPS/JCALL/JScripter

Returns the date-and-time stamp of the specified file.

➘ Sfn is the name of the file.

➚ Ddatetime is the date-and-time stamp of the file. You can use the date-time conversion and formatting routines on the returned value.

GetFileDate, SetFileDate, DateToStr, DateTimeToStr, FormatDateTime

Returns `FALSE` if the file was not found.

FileExists (Sfn) : B

JoPPS/JCALL/JScripter

Checks if a given file exists.
Since JoPPS 3.21 P1 this function works also with wildcards (*,?).

➘ Sfn is the name of the file to check.

➚ Returns `TRUE` if the file exist.

🕮 `DirExists, FileSearch, FilesExist`

FileLength (Dh) : D

JoPPS/JCALL/JScripter

Returns the total length in bytes of a file.

➘ Dh is the filehandle of the file.

➚ The total length of the file in bytes.

FilePos

`FilePos (Dh) : D`

JoPPS/JCALL/JScripter

Returns the current file pointer position of a file.

➘ Dh is the file handle of the file.

➚ The position of the file pointer from the beginning of the file in bytes.

FileLength

`FileSearch (Sspec,Sdirlist) : S`

JoPPS/JCALL/JScripter

Searches for a given file in a list of directories.

➘ Sspec is the name of the file to search for. Sdirlist is the list of subdirectories to search.
➘ The directory paths in Sdirlist must be separated by semicolons.

➚ The returned value is a concatenation of one of the directory paths and the filename, or an empty string if the file could not be located.

Returns an empty string if the file could not be found !

FileExists

`FilesExist (Sfn, Dattr=0) : B`

JoPPS/JCALL/JScripter

Checks if a given file exists. Available in Jopps 2.84 P2

➘ Sfn is the name of the file to check.
This function allows the use of wildcards (*,?).
➘ Dattr is an optional parameter to specify the file attributes for the file to check. The possible values are:

Value	Meaning
`ATTR_READONLY`	Look for files with the ReadOnly attribute set
`ATTR_HIDDEN`	Look for files with the Hidden attribute set
`ATTR_SYSFILE`	Look for files with the System File attribute set
`ATTR_VOLUMEID`	Look for VolumeID files
`ATTR_DIRECTORY`	Look for Directories
`ATTR_ARCHIVE`	Look for files with the Archive attribute set (typical for zip files)
`ATTR_SYMLINK`	
`ATTR_ANYFILE`	Look for any file

➚ Returns `TRUE` if the file exist.

🕮 `DirExists, FileSearch, FileExists`

`FileToStr (Sfn) : S`

JoPPS/JCALL/JScripter

Reads a textfile into a string variable.

➘ Sfn is the name of the textfile to read.

➚ Returns an S-type value holding the contents of the file.

Returns an empty string if the file cannot be read.

🕮 StrToFile

`FileType (Dh) : D`

JoPPS/JCALL/JScripter

Returns the type of a file.

➘ Dh is the filehandle representing the file.

➚ Returns one of the following constants :

Filetype constants	Meaning
`FILETYPE_UNKNOWN`	The type of the file is unknown.
`FILETYPE_DISK`	The file is a disk file.
`FILETYPE_CHAR`	The file is a character file, typically a LPT device or a console.
`FILETYPE_PIPE`	The file is either a named or anonymous pipe.

`FindOverlap (Ssystem1, Sprofile1, Ssystem2, Sprofile2[ ,Dside]) : D`

JoPPS/JCALL/JScripter

Find combination between two profiles.

➘ Ssystem1 is the system code of the first profile
➘ Sprofile1 is the profile code of the first profile
➘ Ssystem2 is the system code of the second profile
➘ Sprofile2 is the profile code of the second profile
➘ Dside: 0 = default overlap, 1 = overlap side I, 2 = overlap side II

➚ Returns the overlap value

`FindShift (Ssystem1, Sprofile1, Ssystem2, Sprofile2[,Drail]) : D`

JoPPS/JCALL/JScripter

Find the shift between an outerframe and sahs profile

➘ Ssystem1 is the system code of the first profile
➘ Sprofile1 is the profile code of the first profile
➘ Ssystem2 is the system code of the second profile
➘ Sprofile2 is the profile code of the second profile
➘ Drail: position sash in outer frame, 0 = default, 1 = rail 1, 2 = rail 2, 3 = rail 3

➚ Returns the shift value

`FLOOR (D) : D`

JoPPS/JCALL/JScripter

Returns the highest integer less than or equal to the given value.

➘ D-type value to be rounded.

➚ Returns the rounded value.

Example:

```
FLOOR (-2.8)     returns -3
FLOOR(2.8)       returns 2
```

`FLOOR(-1.0)` returns -1

📖 `CEIL`

`FlushKbd ()`

JCALL

Empties the keyboard buffer. (standard console input stream)

`KeyPressed, KeyIn`

`FormatDateTime (Sformat,Ddatetime) : S`

JoPPS/JCALL/JScripter

Formats a given date-and-time value using the format string given.

➘ Sformat is the date-and-time format string,
Ddatetime is the date-and-time value. (eg. returned by Now)

Format strings :

c	Displays the date using the ShortDate (Windows regional settings) format, followed by the time in LongTime (Windows regional settings) format. The time is not displayed if the fractional part of the Ddatetime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat).
dddd	Displays the day as a full name (Sunday-Saturday) .
ddddd	Displays the date using the ShortDate (Windows regional settings) format.
dddddd	Displays the date using the LongDate (Windows regional settings) format.
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec).
mmmm	Displays the month as a full name (January-December).
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time using the ShortTime (Windows regional settings) format.
tt	Displays the time using the LongTime (Windows regional settings) format.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case,
and	the result is displayed accordingly.
/	Displays the date separator character.
:	Displays the time separator character.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

➚ The formatted string.

Example :

```
msg := FormatDateTime('"It is today " dddd, mmmm d, yyyy, ',Now(NOW_DATE));
OutputMsg(msg);
```

`DateToStr, TimeToStr, Now`

```
FormatStr (Sformat,D) : S
FormatStr (Sformat,S) : S
```
JoPPS/JCALL/JScripter

Formats a D-type or S-type variable using the format string S.

➘ Sformat is the format string,
➘ The D- or S-type parameter is used as the argument for the formatter.

Format strings :

Format strings contain two types of objects--plain characters and a format specifier. Plain characters are copied verbatim to the resulting string. The format specifier uses the second argument to apply formatting to.
A format specifier can have the following form :
"%" [index ":"] ["-"] [width] ["." prec] type
A format specifier begins with a % character. After the % come the following, in this order :
An optional argument index specifier, [index ":"]
An optional left justification indicator, ["-"]
An optional width specifier, [width]
An optional precision specifier, ["." prec]
The conversion type character, type; which should be compatible with the second argument given :

a) the second argument is of the S-type : **`FormatStr(Sformat,S):S`**

s String.
The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.

b) the second argument is of the D-type : **`FormatStr(Sformat,D):S`**

e Scientific.
The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd".
The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point.
The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string--a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f Fixed.
The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...".
The resulting string starts with a minus sign if the number is negative.
The number of digits after the decimal point is given by the precision specifier in the format string--a default of 2 decimal digits is assumed if no precision specifier is present.
g General.
The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format.
The number of significant digits in the resulting string is given by the precision specifier in the format string--a default precision of 15 is assumed if no precision specifier is present.
Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.
n Number.
The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...".
The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.
m Money.
The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the Windows regional settings.

➚ The formatted string.

```
FtpDownLoad(sUrl,sFolder);
```
JoPPS – V3.33

Function to download a file via FTP

➘	sUrl	location filename
➘	sFolder	location on computer

```
FtpUpLoad(sFilename,sHost,sUser,sPassword,sFolder[,dPort[,dPassive]]);
```
JoPPS – V3.33

Function to upload a file via FTP

➘	sFilename	filename
➘	sHost	servername
➘	sUser	username
➘	sPassword	password
➘	sFolder	location on server
➘	dPort	port
➘	dPassive	passive /active

```
FUN(Dfunid)
```
JoPPS – V2.82

This function is an alias for SelectEditorFunction.

SelectEditorFunction

```
GetActiveProjectIndex () : Dindex
```
JoPPS

Returns the index of the active project in the projectpool.

➚ The index of the active project in the projectpool.

A value of -1 is returned when no project is loaded. (the projectpool is empty)

SetActiveProjectIndex, ProjectCount

🕮 See also *3. Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

```
GetAutoBackup() : Bstate
```
JoPPS 3.28

Returns the “Make backup file before overwritng” setting.

➚	True	activated
➚	False	deactivated

```
GetAutoRecover() : Bstate
```
JoPPS 3.28

Returns the “Restore project automatically” setting.

➚	True	activated
➚	False	deactivated

```
GetAutoSaveInterval() : Ddelay
```
JoPPS 3.28

Returns the “Interval in minutes”

➚	Ddelay	time in minutes

```
GetAutoSaveOnclose() : Bstate
```
JoPPS 3.28

Returns the “Autosave before closing a project” setting.

➚ True activated
➚ False deactivated

`GetAutoSaveOnCreate() : Bstate`
JoPPS 3.28

Returns the “Save new project immediately” setting.

➚ True activated
➚ False deactivated

`GetAutoSaveTimer() : Bstate`
JoPPS 3.28

Returns the “Autosave according to interval of time” setting.

➚ True activated
➚ False deactivated

`GetCalcBehaviour() : Dcalcbehaviour`
JoPPS 3.37

Returns the current J*o*PPS calculation behaviour. The calculation behaviour can be changed by the user through the J*o*PPS Project.Mode menu.

➚ The current calculation behaviour setting. Can be one of the following constants:

Calculation behaviour constants	Meaning
`CALCBEHAVIOUR_USER`	?
`CALCBEHAVIOUR_DEFAULT`	Calculate
`CALCBEHAVIOUR_NOREPORTS`	Calculate (without reports)
`CALCBEHAVIOUR_OFFER`	Calculate (commercial)
`CALCBEHAVIOUR_ORDER`	Calculate (order)
`CALCBEHAVIOUR_PRODUCTION`	Calculate (production)

SetCalcBehaviour

`GetCalcMode() : Dcalcmode`
JoPPS

Returns the current J*o*PPS calculation mode. The calculation mode can be changed by the user through the J*o*PPS Project.Mode menu.

➚ The current calculation mode setting. Can be one of the following constants :

Calculation mode constants	Meaning
`CALCMODE_GROUP`	Calculate current assembly (group) of current project only
`CALCMODE_PROJECT`	Calculate current project only
`CALCMODE_BATCH`	Calculate in batch mode
`CALCMODE_PTABLE`	Calculate in pricetabel mode

Refer to the explanation of the `Calculate` function for a discussion of the calculation mode constants.

🕮 Refer to *3. Using JoPPS-Script in JoPPS* for more information on the calculation mode.

SetCalcMode, Calculate

`GetCalcPerBatchType( ? ) : ?`
JoPPS – V2.82

To be documented.

`GetCurDir() : Sdir`

JoPPS/JCALL/JScripter

Returns the current directory.

➚ Sdir holds the current disk and directory specification.

SetCurDir

`GetCurrentAssembly() : Oassembly`

JoPPS - v2.30

Returns the current assembly. (the assembly being edited in the JoPPS editor)

➚ Oassembly, the current assembly object.

GetCurrentProject

`GetCurrentLanguage( ? ) : ?`

JoPPS – V2.82

To be documented.

`GetCurrentProject() : Oproject`

JoPPS - v2.30

Returns the current project.

➚ Oassembly, the current project object.

GetCurrentAssembly

`GetDatabaseDesc() : Sdesc`

JoPPS

Returns the description of the database in use.

➚ Sdesc is the description of the database package currently being used,
an empty string is returned when no database is selected.
(a database description can be changed using the J*o*PPS Administrator program)

GetDatabaseId, SelectDatabase

`GetDatabaseId() : Sdbid`

JoPPS

Returns the id of the database in use.

➚ Sdbid is the id (up to 8 characters) of the database package currently being used,
an empty string is returned when no database is selected.

GetDatabaseDesc, SelectDatabase

`GetDatabaseVersion() : SdbVersion`

JoPPS – V3.30

Returns the id of the database in use.

🕮 GetDatabaseDesc, SelectDatabase, GetSoftwareVersion

`GetDebug( ? ) : ?`

JoPPS – V2.82

To be documented.

```
GetField( ? ) : ?
```
JoPPS – V2.82

To be documented.

```
GetFileDate(Dh) : Ddatetime
```
JoPPS/JCALL/JScripter

Returns the date-and-time stamp of the specified filehandle.

➘ Dh is the filehandle.

➚ Ddatetime is the date-and-time stamp of the file. You can use the date-time conversion and formatting routines on the returned value.

SetFileDate, FileAge, DateToStr, DateTimeToStr, FormatDateTime

```
GetGUIKind( ? ) : ?
```
JoPPS – V2.82

To be documented.

```
GetHandlePos([Bofs]) : Dpos
```
JoPPS/3.34 P3

The script function ' GetHandlePos ' was provided with an additional argument to indicate whether the handle position/height should be determined or not in function of the specified handle reference:

➘ Bofs = False : handle position relative to sash
➘ Bofs = True : handle position depending on to handle reference

➚ Dpos is the position of the handle

```
GetININum( ? ) : ?
```
JoPPS – V2.82

To be documented.

```
GetIniStr( ? ) : ?
```
JoPPS – V2.82

To be documented.

```
GetLanHint(Dtrans[,Stable]) : S
GetLanHint(Dform,Dcontrol[,Stable]) : S
```
JoPPS

Returns the hint for a given translation code.

➘ Dtrans is a message translation code.
➘ Dform is a form translation code identifying a JoPPS dialog and the
➘ Dcontrol argument is the control translation code identifying a specific control on this form.
➘ The optional Stable argument specifies the translation table to use. By default the translation table associated with the running program is used. (eg. jopps.db when in JoPPS)

➚ The hint associated with the given translation code argument(s) or an empty string if the requested entry was not in the translation database or has no hint.

GetLanText, GetReportLanTag

```
GetLanText(Dtrans[,Stable]) : S
GetLanText(Dform,Dcontrol[,Stable]) : S
```
JoPPS

Returns the text for a given translation code.

- ➘ Dtrans is a message translation code.
- ➘ Dform is a form translation code identifying a JoPPS dialog and the
- ➘ Dcontrol argument is the control translation code identifying a specific control on this form.
- ➘ The optional Stable argument specifies the translation table to use. By default the translation table associated with the running program is used. (eg. jopps.db when in JoPPS)

➚ The text associated with the given translation code argument(s) or an empty string if the requested text was not in the translation database.

Example :

```
OutputMsg("List of possible run-time errors :");
i := 0;
WHILE (i:=i+1) < 51 DO
{
  errmsg := GetLanText(-i,"jscript");
  OutputMsg("error #"+IntToStr(i)+": "+errmsg);
};
```

GetLanHint, GetReportLanTag

GetMachineCnt(?) : ?

JoPPS – V2.82

To be documented.

GetMachineDesc(?) : ?

JoPPS – V2.82

To be documented.

GetMachineName(?) : ?

JoPPS – V2.82

To be documented.

GetMsgPaneMode(?) : ?

JoPPS – V2.82

To be documented.

GetObjFullDesc(Oatom) : S

JoPPS - v2.70

Returns a descriptive text for the atom object given.

Oatom is the atom object.

GetObjShortDesc(Oatom) : S

JoPPS - v2.70

Returns a short text describing the given atom object.

Oatom is the atom object.

GetParam(Sparam[,Sdefault]) : Svalue

JoPPS

Returns the value of a J*o*PPS parameter.

- ➘ Sparam is the name of the parameter.
- ➘ Sdefault is the default value if the parameter is not known to the system.

➚ The value of the requested parameter or the default or an empty string if the parameter does not exists.

Example :

```
progroot := GetParam("PROGRAM_ROOT"); /* do NOT add the % chars!!! */
OutputMsg("%PROGRAM_ROOT%="+progroot);
```

SetParam, InterpreteString

```
GetPRIORPTPATH ( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
GetProdDBVersion( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
GetProdState( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
GetProjectFilename() : S
GetProjectFilename(Sname) : S
GetProjectFilename(Dindex) : S
```

JoPPS

Returns the filename of a project in the projectpool.

If no parameter is specified the current project is assumed,

➘ Sname is the name of a project in the projectpool,
➘ Dindex is a projectpool index.

➚ The fully qualified filename or an empty string if the specified project was not found in the projectpool. (or no projects are open!)

🕮 See also *3. Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

```
GetProjectPool( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
GetReportDesc( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
GetReportLanTag([Dslot]) : Dtrans
```

JoPPS

Returns the translation code for the given report slot

➘ Dslot is the optional slot index. If no parameter is passed the current slot is assumed.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.
GetLanText, GetLanHint

```
GetReportOutputFiles( ? ) : ?
```

JoPPS – V2.82

To be documented.

`GetReportType([Dslot]) : S`

JoPPS

Returns the reporttype of a slot.

➘ Dslot is the optional slot index. If no parameter is passed the current slot is assumed.

➚ Returns "H"=HTML, "T"=Text, "L"=Label and "J"=J*o*PPS-Script or an empty string
if no valid slot index is specified.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

`GetResultLineCount([Dslot]) : D`

JoPPS

Returns the number of text lines in the result of the given report slot.

➘ Dslot is the optional slot index. If no parameter is passed the current slot is assumed.

➚ Returns the number of textlines in the result. Returns 0 if the slot was not updated during the
last report run.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.
`ReportRan, GetResultStr`

`GetResultParam(DparamId) : DSI`

JoPPS

Returns the value of the specified result parameter.

➘ DparamId is the id of the parameter to query.

➚ Returns the value of the specified result parameter.

Parameter ids	Required type	Meaning
`PARAM_INTRO`	`B-type`	The state of the main-intro flag
`PARAM_SUBINTRO`	`B-type`	The state of the sub-intro flag
`PARAM_SUBSUMMARY`	`B-type`	The state of the sub-summary flag
`PARAM_SUMMARY`	`B-type`	The state of the main-summary flag

`SetResultParam`

`GetResultStr([Dslot[,Dline]]) : S`

JoPPS

Returns the result for a given slot.

➘ Dslot is the optional slot index,
➘ Dline is to select a specific line in the result text

➚ The result text or line as a string; an empty string is returned if the slot was not up-to-date.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

Its better to specify a slot index as it is possible there is no current slot.

GetSaveToDisk() : B

JoPPS

Returns the state of the J*o*PPS "SaveToDisk" flag. The SaveToDisk flag controls whether or not J*o*PPS results should be written to disk.

➚ Returns the state of the "SaveToDisk" flag.

Example :

```
IF !GetSaveToDisk() THEN
{
  IF AskYN("SaveToDisk is not active! Activate ?") THEN SetSaveToDisk(TRUE);
};
```

SetSaveToDisk

GetSendToProd(?) : ?

JoPPS – V2.82

To be documented.

GetSoftwareversion( ) : ?

JoPPS 3.00

➚ Returns the software version

GetSubStr(Scollection,Dindex[,Sdelimiter]) : Ssub

JoPPS/JCALL/JScripter

Selects a specified substring out of a given "collection" string, you can specify a delimiter used to separate the strings in the "collection" string. The default delimiter is the comma (',') character.

➘ Scollection is the "collection" string,
➘ Dindex is the substring to return
➘ Sdelimiter is an optional delimiter character.

➚ Ssub is the substring or an empty string if Dindex is beyond the number of substrings in the "collection" string.

Example :

```
days := "Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday";
day := DayOfWeek();
OutputMsg("Today is "+GetSubStr(days,day-1));
```

SubStrCnt

GetSyntax(?) : ?

JoPPS – V2.82

To be documented.

GetUI() : B

JoPPS

Returns the state of the internal UI (userinterface) flag. The UI flag indicates whether or not J*o*PPS should popup the batch- or pricetableparameter dialog prior to start calculations.

➚ The state of the UI flag.

SetUI

`GetURLFile( ? ) : ?`

JoPPS – V2.82

To be documented.

`GetUserDesc() : Sdesc`

JoPPS

Returns the description of the current J*o*PPS user.

➚ Sdesc is the description of the current user.
(a user description can be changed using the J*o*PPS Administrator program)

`GetUserId`

`GetUserId() : Suid`

JoPPS

Returns the id of the current J*o*PPS user.

➚ Suid is the id of the current user (max 8 characters).

`GetUserDesc`

`GetUseScrapMan( ? ) : ?`

JoPPS – V2.82

To be documented.

`HasLicenseOption(Doption) : B`

JoPPS – V3.26

Can be used to verify whether a specific license option is active.

➘ Doption is a number between 19..40

➚ Returns `TRUE` if the specified option is active

`HasResult([Dslot]) : B`

JoPPS

Returns if the specified report slot is up-to-date, thus contains a valid result.

➘ Dslot is the slot index, if not Dslot argument is passed the current slot is assumed.

➚ Returns `TRUE` is the specified slot is up-to-date.

Its better to specify a slot index as it is possible there is no current slot.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

`HTMLStr( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`HTMLToNormalStr(Stext[,Bblank],Bstrip]]) :`

JoPPS/JCALL/JScripter – V2.82, V3.33P4

➘ Stext is the text to edit
➘ Bblank replace *;nbsp* by spaces ?
➘ Bstrip remove spaces from text ?

`IF( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`IIF( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`ImportFile( ? ) : ?`

JoPPS – V2.82

To be documented.

`ImportFilterInstalled( ? ) : ?`

JoPPS – V2.82

To be documented.

`InstalledFrom() : Sdir`

JoPPS/JCALL/JScripter

Returns the location from where this version of J*o*PPS (eg. and thus the interpreter) was installed.

➚ SDir is the installation path.

InstalledIn

`InstalledIn() : Sdir`

JoPPS/JCALL/JScripter

Returns the location where the J*o*PPS executable files reside. This information is taken from the windows registry where JoPPS keeps its last startup location.

➚ SDir is the J*o*PPS program path.

InstalledFrom

`InstallImportFilter( ? ) : ?`

JoPPS – V2.82

To be documented.

`InstallPlugin( ? ) : ?`

JoPPS – V2.82

To be documented.

`InterpreteString(Sparam) : Sresult`

JoPPS

Interpretes the given string by substituting all J*o*PPS parameters in the input string by their value.

➘ Sparam is the string to interprete containing none, one or multiple J*o*PPS parameters.
The JoPPS parameters should be enclosed by % characters.

➚ Returns the string with all known parameters substituted.

If used in HTML scripts the J*o*PPS report generator will automatically substitute all known J*o*PPS parameters it encounters before the InterpreteString function is executed !

GetParam, SetParam

`IntToHex(D,Dwidth) : S`

JoPPS/JCALL/JScripter - v2.0

Converts an integer into a string holding its hexidecimal value.

➘ D is the integer value to be converted, Dwidth is the width of the returned string.

➚ The string holding the intergers hexidecimal value.

NumToStr, StrToNum, IntToStr, IntToStr0

`IntToStr(D) : S`

JoPPS/JCALL/JScripter

Converts an D-type value into a S-type value assuming D represents an integer value.

➘ D is the integer value to be converted.

➚ The string representing the integer value.

NumToStr, StrToNum, IntToStr0

`IntToStr0(D, Dwidth) : S`

JoPPS/JCALL/JScripter - v2.0

Converts an D-type value into a S-type value assuming D represents an integer value. The resulting string is left-padded with 0 characters.

➘ D is the integer value to be converted,
➘ Dwidth is the length of the returned string. The string is left-padded with 0 characters if needed.

➚ The left-padded string representing the integer value.

NumToStr, StrToNum, IntToStr

`IsConsole( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`IsConstant( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`IsFunction( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`IsIDispatch(SDIO) : B`

JoPPS/JCALL/JScripter

Returns if the variable passed is an I-type variable

Argument can be of any type.

➚ Returns `TRUE` if the passed argument is of the I-type.

IsNumber, IsString, IsObject

`IsNill( ? ) : ?`

JoPPS/JCALL/JScripter – V2.82

To be documented.

`IsNumber(SDIO) : B`

JoPPS/JCALL/JScripter

Returns if the variable passed is a D-type variable

Argument can be of any type.

↗ Returns `TRUE` if the passed argument is of the D-type.

IsString, IsIDispatch, IsObject

`IsObject(SDIO) : B`

JoPPS/JCALL/JScripter - v2.0

Returns if the variable passed is an O-type variable

Argument can be of any type.

↗ Returns `TRUE` if the passed argument is of the O-type.

IsString, IsNumber, IsIDispatch

`IsPluginInstalled( ? ) : ?`

JoPPS – V2.82

To be documented.

`IsProductionRunning( ? ) : ?`

JoPPS – V2.82

To be documented.

`IsReportTagged([Dslot]) : B`

JoPPS

Returns whether or not the specified report slot is tagged.

↘ Dslot is the optional slot index. If no argument is specified the current slot is assumed.

↗ Returns `TRUE` if the specified slot index is tagged.

Its better to specify a slot index as it is possible there is no current slot.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

`IsRunning() : B`

JoPPS

Can be used to verify whether or not J*o*PPS is busy running its calculations or updating reports.

↗ Returns `TRUE` if J*o*PPS is busy.

`IsString(SDIO) : B`

JoPPS/JCALL/JScripter

Returns if the passed variable is a S-type variable.

Argument can be of any type.

↗ Returns `TRUE` if the passed argument is of the S-type.

IsNumber, IsIDispatch

`IsTimerLicense() : B`

JoPPS/JCALL/JScripter

Can be used to verify whether or not the license is a timer license

➚ Returns `TRUE` if timer license.

IsDealerLicense

`JoPPSDirect( ? ) : ?`

JoPPS – V2.82

To be documented.

`KeyIn() : Dkey`

JCALL

Waits for a keypress and returns the ascii code for the key pressed.

➚ Dkey is the ascii code for the key pressed.

KeyPressed

`KeyPressed() : B`

JCALL

Checks if a key is waiting in the input buffer.

Returns `TRUE` if a key is pressed, `FALSE` if not.

Pause, FlushKbd, KeyIn

`Kill(Svarname)`

JoPPS/JCALL/JScripter

Disposes a specified variable. Use kill on a variable when you do not longer need it.

➘ Svarname is the name of the variable to free, if Svarname is an empty string <u>all</u> variables are freed.

START

`LicenseId() : S`

➚ Returns the license key.

`LicenseName() : S`

➚ Returns the name.

`LineIn() : S`

JCALL

Reads a line of text from the standard input stream.

➚ Returns the string read from the standard input stream.

CharIn

`LineOut(S)`

JoPPS/JCALL

Writes a line of text followed by an CR/LF pair to the standard output stream.

➘ S is the text to output.

TextOut, CharOut

LN(D) : D

JoPPS/JCALL/JScripter

Returns the natural logarithm of the argument.

LOG

LOG(D) : D

JoPPS/JCALL/JScripter

Calculates the log base 10 of the argument.

LN

Login([Suid[,Spassword]]) : S

JoPPS

Performs a user login operation. Pops up the J*o*PPS login dialog if not all required parameters are specified.

➘ Suid is the default userid,
➘ The optional argument Spassword specifies the password for user Suid.

The Spassword argument is not required if the user has no password set.

➚ Returns the id of the current user after the login operation.

SelectDatabase

Lower(S) : S

JoPPS/JCALL/JScripter

Converts a string to lowercase.

➘ S is the string to be converted.

➚ The string converted to lowercase.

Upper

LTrim (S) : S

JoPPS/JCALL/JScripter

Deletes leading spaces and control characters from a string.

➘ String to trim.

➚ The "trimmed" string.

RTrim, Trim

MakeDir (Sdir) : B

JoPPS/JCALL/JScripter

Creates a new subdirectory.

➘ Sdir is the name for the new subdirectory.

➚ Returns `TRUE` if successful.

RemoveDir, SetCurDir, GetCurDir

`MAX (D1,D2[,D3..]) : D`

JoPPS/JCALL/JScripter

Returns the greater of two or more numeric values.

The numeric values to compare.

➚ The greater of the two given values.

Example :

```
a := 10;
b := 20;
MsgBox(Max(a,b)," is greater then ",Min(a,b));
```

MIN

`MB(SD)`

JoPPS/JCALL/JScripter – V2.82

This function is an alias for MsgBox.

MsgBox

`ME(SD)`

JoPPS/JCALL/JScripter – V2.82

This function is an alias for MsgErr.

MsgErr

`MIN (D1,D2[D3..]) : D`

JoPPS/JCALL/JScripter

Returns the lesser of two or more numeric values.

The numeric values to compare.

➚ The lesser of the two given values.

MAX

`Month ([Ddatetime]) : Dmonth`

JoPPS/JCALL/JScripter

Returns the month.

➘ If no parameter is given todays date is used as input, the optional Ddatetime argument specifies an alternative date.

➚ Dmonth is the month, a number in the range 1..12.

Now, Day, Year, DayOfWeek, WeekOfYear, DayOfYear

`MoveFile (Sfrom,Sto) : B`

JoPPS/JCALL/JScripter

Moves the file Sfrom to Sto.

➘ Sfrom is the file to move, Sto defines the location and name for the file

➚ Returns `TRUE` when the operation was successful.

🕮 `CopyFile, CopyFileTo, RenameFile`

```
MsgBox (SD)
```
JoPPS/JCALL/JScripter

Pops up a message box displaying the result of the concatenation of the parameters passed.

➘ A multiple of S-type and D-type parameters can be passed. Their string representations are concatenated and displayed in the message box.

MsgBox2, MsgErr, MsgErr2

```
MsgBox2 (Stitle,SD)
```
JoPPS/JCALL/JScripter - v2.0

Pops up a message box displaying the result of the concatenation of the parameters passed.

➘ Stitle is the messagebox caption followed by a multiple of S-type and D-type parameters can be passed. Their string representations are concatenated and displayed in the message box.

MsgBox2, MsgErr, MsgErr2

```
MsgErr (SD)
```
JoPPS/JCALL/JScripter

Pops up a error message box displaying the result of the concatenation of the parameters passed.

➘ A multiple of S-type and D-type parameters can be passed. Their string representations are concatenated and displayed in the message box.

MsgErr2, MsgBox, MsgBox2

```
MsgErr2 (Stitle, SD)
```
JoPPS/JCALL/JScripter - v2.0

Pops up a error messagebox displaying the result of the concatenation of the parameters passed.

➘ Stitle is the messagebox caption followed by a multiple of S-type and D-type parameters can be passed. Their string representations are concatenated and displayed in the message box.

MsgErr2, MsgBox, MsgBox2

```
MsgPaneCount () : D
```
JoPPS

Returns the number of messages in the J*o*PPS message pane.

➚ The number of messages in the message pane. (both messages <u>and</u> errormessages)

MsgPaneErrCount
🕮 see also *3. Using JoPPS-Script in JoPPS*

```
MsgPaneErrCount () : D
```
JoPPS

Returns the number of errormessages in the J*o*PPS message pane.

➚ The number of errormessages in the message pane. (only the errormessages)

MsgPaneCount
🕮 see also *3. Using JoPPS-Script in JoPPS*

```
MsgPaneGet (Dmsgindex) : Smsg
```
JoPPS

Returns a message from the J*o*PPS message pane

➘ Dmsgindex is the message index (eg. the line) of the message in the message pane.

➚ The message at line Dmsgindex in the message pane.

MsgPaneGetErrCode
🕮 see also *3. Using JoPPS-Script in JoPPS*

`MsgPaneGetErrCode (Dmsgindex) : Derrcode`
JoPPS

Returns an errorcode from the J*o*PPS message pane.

➘ Dmsgindex is the message index (eg. the line) of the message in the message pane.

➚ The errorcode of the message at line Dmsgindex in the message pane,
if the errorcode equals 0 the message at index Dmsgindex is a message instead of an errormessage.

MsgPaneGet
🕮 see also *3. Using JoPPS-Script in JoPPS*

`MsgPaneIsOpen () : B`
JoPPS

Returns the state of the message pane in J*o*PPS.

➚ Returns `TRUE` is the J*o*PPS message pane is currently open, otherwise `FALSE` is returned.

OpenMsgPane, CloseMsgPane, OutputMsg
🕮 see also *3. Using JoPPS-Script in JoPPS*

`NetSend( ? ) : ?`
JoPPS – V2.82

To be documented.

`NewBatchName() : Sname`
JoPPS

➚ Sname is the generated batch name on basis of the mask defined in the JoPPS Administrator

`NewBiColour(Odialog) : Scolor`
JoPPS – V3.27

➘ Odialog is the Bi-Colour wizard dialog object

➚ New Bi-Colour code (blank if canceled)

NewProjectName() : Sname

JoPPS

➚ Sname is the generated project name on basis of the mask defined in the JoPPS Administrator

NoBackSlash (Spath) : S

JoPPS/JCALL/JScripter

Deletes the trailing backslash from a given path specification.

➘ Spath is a given path specification (eg. c:\joppswin\)

➚ S is the path specification without a trailing backslash. (eg. c:\joppswin)

Example :
```
dxfview := NoBackSlash(InstalledIn())+"\dxfview.exe";
IF FileExists(dxfview) THEN
  RunProgram(dxfview)
ELSE
  Fatal("The executable <"+dxfview+"> does not exist!");
```

If the path specification does not end in a backslash the input string is returned unchanged.

Now ([Dhow]) : Ddatetime

JoPPS/JCALL/JScripter

Returns the current date and time.

➘ The optional argument Dhow can be used to specify if one only wants the time or date part to be returned.

Dhow	Meaning
`NOW_DATETIME`	Return current date and time (default)
`NOW_DATE`	Return current date only
`NOW_TIME`	Return current time only

➚ The current date and or time as a floating point value.
24h corresponds to a value of 1, thus 12h equals 0.5 and so on..

Example :

```
time := Now(NOW_TIME);
IF time < 0.5 THEN
  msg := "Goodmorning"
ELSE IF time < 0.75 THEN
  msg := "Good afternoon"
ELSE
  msg := "Good evening";
```

Day, Month, Year, DayOfWeek, WeekOfYear, DayOfYear, DateToStr

NumToStr (Dnum[,Dwidth[,Ddecimals]]) : S

JoPPS/JCALL/JScripter

Converts a D-type variable into a S-type variable.

➘ Dnum is the number to convert,
➘ Dwidth is the total number of positions
➘ Ddecimals is the number of decimals used.

➚ The converted number as a string.

IntToStr, IntToStr0, StrToNum

ObjToInt (O) : D

JoPPS - v2.50

Converts an object (its pointer value) into its numeric representation.

➘ The object reference.

➚ The object address (pointer value) as a number.

🕮 `AtomToObj`

This function is intended to be used in specific JoPPS result reports.

`OM(S [, D])`

JoPPS/JCALL – V2.82

This function is an alias for OutputMsg.

`OutputMsg`

`OpenFile (Sfn[,Dopenmode]) : Dh`

JoPPS/JCALL/JScripter

Opens a file.

➘ Sfn is the filename,
➘ Dopenmode is an optional argument specifying how the file is opened, the default openmode is OPENMODE_READWRITE.

You can use the following constants as arguments to the Dopenmode parameter :

File open mode constants	Meaning
`OPENMODE_READ`	Open for read access only
`OPENMODE_WRITE`	Open for write access only
`OPENMODE_READWRITE`	Open for read and write access
`OPENMODE_SHAREEXCLUSIVE`	Read and write access for others is denied
`OPENMODE_SHAREDENYWRITE`	Write access for others is denied
`OPENMODE_SHAREDENYREAD`	Read access for others is denied
`OPENMODE_SHAREDENYNONE`	Allows full access for others

➚ Returns a valid filehandle Dh or a value of -1 if failed.

`OpenRead, OpenWrite, CloseFile`

Close the file using the `CloseFile` function when you are done using it.

`OpenMsgPane()`

JoPPS/JCALL/JScripter

Opens the JoPPS message pane. The message pane is the window where all J*o*PPS messages are shown.

`CloseMsgPane, ClearMsgPane, OutputMsg`

🕮 see also *3. Using JoPPS-Script in JoPPS*

`OpenRead (Sfn) : Dh`

JoPPS/JCALL/JScripter

Opens a file for read access only.

➘ Sfn is the filename.

➚ Returns a valid filehandle Dh or -1 if failed.

`OpenWrite, OpenFile, CloseFile`

Close the file using the `CloseFile` function when you are done using it.

`OpenWrite (Sfn) : Dh`

JoPPS/JCALL/JScripter

Opens a file for write access only.

➘ Sfn is the filename.

➚ Returns a valid filehandle Dh or -1 if failed.

OpenRead, OpenFile, CloseFile

Close the file using the `CloseFile` function when you are done using it.

`ORD (Schar) : D`

JoPPS/JCALL/JScripter

Returns the ordinal value of a character.

➘ Schar is a string holding at least 1 character.

➚ Returns the ordinal value of the character. (its ascii code)

CHR

`OutputMsg (S[,D])`

JoPPS/JCALL

Outputs a message to the J*o*PPS message pane. The message pane is the window where all J*o*PPS messages are shown. If the message pane is currently closed it will be opened automatically.

➘ S is the message to display,
➘ D is an optional errorcode. (= 0 means it is just a message, <> 0 means an errormessage)

🕮 see also *3. Using JoPPS-Script in JoPPS*

OpenMsgPane, CloseMsgPane, ClearMsgPane

`Pause ()`

JCALL

Pauses the script execution untill the user presses a key.

Example :

```
LineOut("Press a key to continue..");
FlushKbd();
repeat until KeyPressed();
LineOut("A key was pressed..");
LineOut("Press once again a key to continue..");
Pause();
LineOut("A key was pressed..");
```

KeyPressed

`PC (Bforced) : Bclosed`

JoPPS – V2.82

This function is an alias for ProjectClose.

ProjectClose

`PerformAction (Scode) : S`
`PerformAction (Scode) : D`

JoPPS - v2.30

Triggers the action with code Scode. The result set by the action is returned.

➘ Scode is the code of the action to trigger.

➚ Returns the result as set by the action.

`PerformEditorFunction (Dfunid)`

JoPPS 3.20

Perform a specific editorfunction.

➘ Dfunid denotes the function to be perform

funid	*editorfunction*
-21000	*Add opening*
-21001	*Remove atom*
-21002	*Redefine opening*
-21003	*Remove handle*
-21004	*Remove sill*
-21020	*Remove extra profile*
-21005	*Remove finishing*
-21006	*Remove accessories*
-21007	*Remove enforcement*
-21008	*Remove filling*
-21009	*Remove ventilator*
-21419	*Remove ventilator (on a T-mullion)*
-21010	*Remove georgian crosses*
-21011	*Remove glazing bead*
-21012	*Remove profile*
-21019	*Remove operation*
-21013	*Add vent*
-21021	*Add vent (2)*
-21014	*Remove framepart*
-21015	*Remove element*
-21016	*Remove filling*
-21017	*Remove ventpart*
-21018	*Remove vent*
-21100	*Add element*
-21101	*Add frameelement*
-21102	*Add ventelement*
-21103	*Add segment*
-21104	*Add T-mullion*
-21122	*Add T-mullion (framelevel)*
-21123	*Add T-mullion (ventlevel)*
-21106	*Add horizontal T-mullion*
-21107	*Add vertical T-mullion*
-21108	*Add fictive*
-21109	*Add general*
-21110	*Add internal*
-21111	*Add ventpart*
-21112	*Redefine ventpart*
-21120	*Add framepart*
-21126	*Add framepart (2)*
-21121	*Redefine framepart*
-21113	*Add splitter*
-21114	*Add origin*
-21115	*Add closure*
-21116	*Add broker*
-21117	*Add relative*
-21118	*Set X reference*
-21119	*Set Y reference*
-21203	*Add handle*
-21204	*Add sill*
-21214	*Add extra profile*
-21205	*Add finishing*
-21206	*Add accessories*
-21207	*Add enforcement*
-21208	*Add filling*
-21209	*Add ventilator*
-21418	*Add ventilator (on T-mullion)*

-21210	*Add georgian crosses*
-21211	*Add glazing bead*
-21212	*Add profile*
-21213	*Add operation*
-21300	*Move T-mullions*
-21301	*Align T-mullions*
-21302	*Align to reference*
-21303	*Align T-mullion in corner*
-21304	*Reshape framecorner*
-21305	*AdjustStijlen*
-21306	*Remove T-mullions*
-21307	*Align vertical*
-21308	*Align horizontal*
-21309	*Align equal*
-21310	*Edit node*
-21311	*Merge*
-21312	*Split*
-21313	*Insert splitter*
-21314	*Delete splitter*
-21400	*Add vertical*
-21401	*Add horizontal*
-21402	*Exchange colours*
-21403	*Static Ix*
-21404	*Static Iy*
-21448	*Select section*
-21405	*Add section*
-21406	*Remove section*
-21449	*Print sections*
-21407	*List sections*
-21411	*Set section*
-21408	*Swap ratio*
-21409	*Offset*
-21447	*Mirror*
-21410	*Calculate weight*
-21412	*Make vent*
-21413	*Add plint*
-21414	*General selection*
-21415	*Change measurements*
-21124	*Store frame*
-21125	*Store vent*
-21420	*Select profielen*
-21421	*Select versterkingen*
-21422	*Select vullingen*
-21423	*Select georgian crosses*
-21424	*Select finishes*
-21425	*Select glazing bead*
-21426	*Select ventilator*
-21417	*Select ventilator (on T-mullion)*
-21427	*Select functional accessories*
-21428	*Select extra accessories*
-21429	*Select operations*
-21430	*Select model*
-21431	*Select sill*
-21432	*Select ventpart*
-21433	*Select vent*
-21446	*Select extra profiles*

SelectEditorFunction

PersistAutoSave() : BState

JoPPS – v3.28

Make the Autosave modifications permanent.

➚ Returns TRUE if successful, FALSE if not

PI () : D

JoPPS/JCALL/JScripter

Returns the value of PI. (e.g. the number 3.1415926535897932385)

PictureHeight(sFile[,nResol])

JoPPS – v3.38 P5

nResult:= PictureHeight(sFile[,nResol]);

➘ sFile is the image file name
➘ nResol is the image resolution (default 120dpi)

➚ nResult is the height of the image

If the image is a DXF the dimensions will be reported in "mm", in the case of a BMP or EMF in "pixels".

PictureWidth(sFile[,nResol])

JoPPS – v3.38 P5

nResult:= PictureWidth(sFile[,nResol]);

➘ sFile is the image file name
➘ nResol is the image resolution (default 120dpi)

➚ nResult is the width of the image

If the image is a DXF the dimensions will be reported in "mm", in the case of a BMP or EMF in "pixels".

PluginHasRoutine(?) : ?

JoPPS – V2.82

To be documented.

PluginLoaded(?) : ?

JoPPS – V2.82

To be documented.

PluginMenuClick(?) : ?

JoPPS – V2.82

To be documented.

PN ([Sfn[,Sdesc[,SParams]]]) : B

JoPPS – V2.82

This function is an alias for ProjectNew.

ProjectNew

PO ([Sfn]) : B

JoPPS – V2.82

This function is an alias for ProjectClose.

ProjectClose

PrintResult ([Bui,Doption])

JoPPS

Prints results.

➘ If Bui is TRUE the JoPPS print dialog opens, If Bui is FALSE printing starts immediately without user interaction.
➘ Doption determines what to print, it can be one of the following constants:

Print option constants	Meaning
`PRINT_CURRENT`	Print result of report slot being viewed.
`PRINT_LABELS`	Print all label slots
`PRINT_ALL`	Print all report slots currently up-to-date.

`PrintStr (S) : B`

JoPPS

Prints the given text to the current print.

➘ S is the text to print.

➚ Returns TRUE if successful, FALSE if not.

`ProjectClose ([Bforced[,bVerbose]]) : Bclosed`

Argument ***[bVerbose]*** sinds JoPPS **3.35**

Closes the current project. The project is removed from the projectpool.

➘ Bforced indicates if the user is allowed to cancel to operation, if `TRUE` the project is always closed even if it is not saved
➘ bVerbose : if the optional argument is `TRUE` (default) the user has to confirm warnings by means of a dialog box.

➚ Returns `TRUE` if the current project was closed by the operation.

`ProjectNew, ProjectOpen, ProjectCount`
🕮 See also *3. Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

ProjectCount () : D
JoPPS

Returns the number of open projects.

➚ Returns the size of the projectpool, thus the number of open projects.

📖 See also 3. *Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

SetActiveProjectIndex, GetActiveProjectIndex

ProjectNew ([Sfn[,Sdesc[,SParams]]]) : B
JoPPS

Adds a new project to the projectpool. The new project becomes the active project. The "new project" dialog is shown if the internal UI flag is `TRUE`, else it is not, and the project is added to the projectpool immediately.

➘ Sfn is the name for the new project. If an empty string is specified a default filename is assigned.
➘ Sdesc is the short description of the project.
➘ SParams is a collection string with default parameters specifying client code, system, filling and finishing codes. (e.g. "TECHWIN,TS50,GL20,51,51,51") Specify an empty string to keep a default value; for example if you only want to override the default filling code you could specify : "TECHWIN,,GL22".

➚ Returns `TRUE` if successful.

Example :

```
SetUI(FALSE);
ProjectNew("my project","","TECHWIN");
```

📖 The following example creates a new project without showing the "new project" dialog :
📖 See also 3. *Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

ProjectClose, ProjectOpen, ProjectCount

A valid clientcode is required when the internal UI flag is `FALSE`.

ProjectOpen ([Sfn[,Dwarn]]) : B
JoPPS

Opens a project. The project is added to the projectpool.

➘ Sfn is the filename of the project file (The JP extension is assumed),
if no filename is specified the windows fileopen dialog will popup allowing the user to pick a file manually.
➘ Dwarn specifies whether generation of warnings messages is enabled or disabled
if no warn parameter is specified warnings will be generated by default

➚ Returns `TRUE` if successful, `FALSE` if not.

ProjectClose, ProjectNew, ProjectCount
📖 See also 3. *Using JoPPS-Script in JoPPS* for a discussion about the J*o*PPS projectpool.

`ProjectSave ([Sfn[,bVerbose]]) : B`
JoPPS

Saves the current project.

Sfn is an optional alternative filename for the project, by default the current project name is used.
If the optional argument bVerbose is `TRUE` the user has to confirm warnings by means of a dialog box.

➚ Returns `TRUE` if the file was saved, `FALSE` if it was not.

Existing files are overwritten without notice. Use `GetProjectFilename` and `FileExists` to ensure no existing files are overwritten.

ProjectSaveAs

`ProjectSaveAs ([Sfn[,bVerbose]]) : B`
JoPPS

Saves the current project under a new name. The windows SaveAs dialog is used to let the user specify the new name.

➘ Sfn is the new filename for the project.
➘ If the optional argument bVerbose is TRUE the user has to confirm warnings by means of a dialog box.

➚ Returns `TRUE` if the file is saved, `FALSE` if it is not.

The ProjectSaveAs function does not overwrite existing files without confirmation.

ProjectSave

`ProjectSave ([Sfn[,bVerbose[,bArchive[,bReserve]]]) : B`
JoPPS

Saves the current project.

➘ Sfn is an optional alternative filename for the project, by default the current project name is used.
➘ If the optional argument bVerbose is TRUE the user has to confirm warnings by means of a dialog box.
➘ If the optional argument bArchive is TRUE the project AND the data is saved
➘ If the optional argument bReserve is TRUE the project is saved in the current state

➚ Returns `TRUE` if the file was saved, `FALSE` if it was not.

Existing files are overwritten without notice. Use `GetProjectFilename` and `FileExists` to ensure no existing files are overwritten.

ProjectSaveAs

`ProjectSaveAs ([Sfn[,bVerbose[,bArchive[,bReserve]]]) : B`
JoPPS

Saves the current project under a new name. The windows SaveAs dialog is used to let the user specify the new name.

➘ Sfn is the new filename for the project.
➘ If the optional argument bVerbose is TRUE the user has to confirm warnings by means of a dialog box.
➘ If the optional argument bArchive is TRUE the project AND the data is saved else only the project is saved (default)
➘ If the optional argument bReserve is TRUE the project is saved in the current state

➚ Returns `TRUE` if the file is saved, `FALSE` if it is not.

ProjectSave

The ProjectSaveAs function does not overwrite existing files without confirmation.

`PS ([Sfn]) : B`

JoPPS/JCALL – V2.82

This function is an alias for ProjectSave.

ProjectSave

`Read (Dh,Dlen) : S`

JoPPS/JCALL/JScripter

Reads a specific number of bytes from a file.

➘ Dh is the filehandle,
➘ Dlen is the number of bytes to read.

➚ Returns a S-type variable holding the bytes read.

Use `StrSize` to determine the number of bytes actually read.

`ReadLn (Dh) : S`

JoPPS/JCALL/JScripter

Reads a line of text from a textfile.

➘ Dh is the filehandle.

➚ Returns the line read from the textfile.

Use the `EOF` function to check if reading past the end of the file.

`ReadStr (Dh) : S`

JoPPS/JCALL/JScripter

Reads a part of a textfile into a string, reading starts at the current file pointer position and up to the end of the file.

➘ Dh is the filehandle

➚ Returns a string holding the contents read.

FileToStr

`Recode (SstartCode[,bVerbose]) : B`

JoPPS - v2.81 P4

Recodes assemblies for current project.

➘ SstartCode specifies first code of recoding sequence assemblies.
➘ If the optional argument bVerbose is TRUE the user has to confirm the action by means of a modal dialog box.

➚ Returns `TRUE` is assemblies recoded, `FALSE` if not.

`RefreshAll([bReset[,bRedraw]]): B`

JoPPS – v3.24

Refreshes the GUI of JoPPS

➘ bReset reset editor functions
➘ bRedraw redraws editor

`RefreshFillings ([bReset]) : D`

JoPPS – V3.34

Refreshes project fillings for current project.

➘ bReset

Default value for the argument bReset = False
If bReset = False the price information is not included if a 'Dealer' version,
if bReset = True the price information is nevertheless included in 'Dealer' version.

➚ Returns `TRUE` if successful, `FALSE` if not.

`RefreshFinishes ([bReset]) : D`
JoPPS – V3.34

Refreshes project finishes for current project.

➘ bReset

Default value for the argument bReset = False
If bReset = False the price information is not included if a 'Dealer' version,
if bReset = True the price information is nevertheless included in 'Dealer' version.

➚ Returns `TRUE` if successful, `FALSE` if not.

`RefreshPriceBlocks () : D`
JoPPS – V2.82

Refreshes priceblocks for current project.

➚ Returns `TRUE` if successful, `FALSE` if not.

`RefreshJobs ([Breset[,Bverbose]]) : Bresult`
JoPPS – V3.30

Refreshes jobs

➘ Breset refreshes the jobs
➘ Bverbose show dialog

➚ Bresult returns the status

`Regen (Oatom) : B`
JoPPS - v2.70

Regenerates math atomdata.

➘ Oatom is the atom object to regenerate.

`RemoveDir (Sdir) : B`
JoPPS/JCALL/JScripter

Removes a subdirectory.

➘ Sdir is the name of the subdirectory to remove.

➚ Returns `TRUE` is successful, `FALSE` if not.

The subdirectory must be empty or `RemoveDir` will fail.

`RenameFile (Sfrom,Sto) : B`
JoPPS/JCALL/JScripter

Renames a file.

➘ Sfrom is the file to rename, Sto is the new name.

➚ Returns `TRUE` if successful, `FALSE` if not.

🕮 `CopyFileTo, CopyFile, MoveFile`

`REPLACEACCESSORYVARIETY (sOld, sNew[, bVerbose])`

JoPPS 3.27

Find and replace variety for accessorys

➘ sOld is the old variety code for accessory
➘ sNew is the new variety code for accessory
➘ bVerbose error messages (default not)

`REPLACEDATADLG (nDlg, oAtom, bVerbose[, …])`

JoPPS 3.27, 3.33

Launch 'Find and replace' dialog visible or invisible

➘ nDlg determines which find and replace dialog to use

DLG_FINISH	variety
DLG_SYSTEM	system
DLG_PROFILE	profiles
DLG_GLAZINGBEAD	glazingbead
DLG_REINFORCEMENT	inforcement
DLG_ACCESSORY	accessory(sets)
DLG_FILLING	filling
DLG_FINISHES	finishing
DLG_JUNCTION	junction set

➘ oAtom determines the start object from where replacements have to be made
➘ bVerbose determines whether the dialogue is shown (True), or not shown (False)
➘ The remaining parameters are depending on the "Find and Replace" dialog, (*) not required:

Parameters for System		
1	DLG_SYSTEM	constant
2	start object	object
3	show dialogue	boolean
4	Find	string
5	find a specific system	boolean
6	replace by	string
7	make default	boolean (*)

Parameters for Finish		
1	DLG_FINISH	constant
2	start object	object
3	show dialogue	Boolean
4	Find	String
5	find a specific finish	Boolean
6	replace by	String
7	profiles	boolean (*)
8	glazing bead	boolean (*)
9	reinforcement	boolean (*)
10	accessories	boolean (*)
11	glazing	boolean (*)
12	window finish	boolean (*)

13	make default	boolean (*)
14	replace for	constant (*)
		LEVEL_ALL
		LEVEL_FRAME
		LEVEL_VENT
15	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT

Parameters for glazing		
1	DLG_FILLING	Constant
2	start object	Object
3	show dialogue	Boolean
4	find	String
5	find a specific filling	Boolean
6	replace by	String
7	replace bij glazing defined	boolean (*)
8	make default	boolean (*)
9	variety	string (*)
10	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
11	price information	constant (*)
		PRICE_INCLUDED
		PRICE_CHARGED
		PRICE_PRICE
		PRICE_OPTION
		PRICE_WRITE
		PRICE_REPORT
		PRICE_EXTRA
12	oversized	boolean (*)
		OVERSIZED_NONE
		OVERSIZED _BOTH
		OVERSIZED _INSIDE
		OVERSIZED _OUTSIDE
13	difficulty degree	constant (*)
14	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH
		ROUTE_DIRECT
15	Filling function	constant (*)
		FILLING_NONE
		FILLING_GLAZING
		FILLING_COVER

		FILLING_PANEL
		FILLING_FLAT
		FILLING_DIAMOND
		FILLING_GRILL
16	Angle	boolean (*)
17	corrections	boolean (*)

Parameters for profile		
1	DLG_PROFILE	Constant
2	start object	Object
3	show dialogue	Boolean
4	find system	String
5	find specific system	Boolean
6	find profile	String
7	find a specific profile	boolean (*)
8	replace by system	String
9	replace by profile	String
10	replace by the profile defined	boolean (*)
11	finish	String
12	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
13	junction	constant (*)
		JUNCTION_MITRE
		JUNCTION_CONTINUE
		JUNCTION_STOP
14	connection	constant (*)
		CONNECTION_CRIMP
		CONNECTION_SCREW
		CONNECTION_CLAMP
15	price information	constant (*)
		PRICE_INCLUDED
		PRICE_CHARGED
		PRICE_PRICE
		PRICE_OPTION
		PRICE_WRITE
		PRICE_REPORT
		PRICE_EXTRA
16	code	boolean (*)
17	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH

Parameters for finishes		
1	DLG_FINISHES	Constant
2	start object	Object
3	show dialogue	Boolean
4	find	String
5	find specific finish	Boolean
6	replace by	String
7	replace by the finish defined	boolean (*)
8	make default	boolean (*)
9	finish	string (*)
10	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
11	price information	constant (*)
		PRICE_INCLUDED
		PRICE_CHARGED
		PRICE_PRICE
		PRICE_OPTION
		PRICE_WRITE
		PRICE_REPORT
		PRICE_EXTRA
12	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH

Parameters for accessories (set)		
1	DLG_ACCESSORY	Constant
2	start object	Object
3	show dialogue	Boolean
4	find	String
5	find specific accessories (set)	Boolean
6	replace by	String
7	replace by the accessories (set) defined	boolean (*)
8	make default	boolean (*)
9	finish	string (*)
10	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
11	price information	constant (*)
		PRICE_INCLUDED
		PRICE_CHARGED
		PRICE_PRICE
		PRICE_OPTION
		PRICE_WRITE
		PRICE_REPORT
		PRICE_EXTRA
12	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH

Parameters for reinforcement		
1	DLG_ REINFORCEMENT	constant
2	start object	object
3	show dialogue	boolean
4	find	string
5	find specific reinforcement	boolean
6	replace by	string
7	replace by the reinforcement defined	boolean (*)
8	make default	boolean (*)
9	finish	string (*)
10	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
11	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH

Parameters for glazingbead		
1	DLG_GLAZINGBEAD	constant
2	start object	object
3	show dialogue	boolean
4	find	string
5	find specific glazingbead	boolean
6	replace by	string
7	replace by the glazingbead defined	boolean (*)
8	make default	boolean (*)
9	finish	string (*)
10	colour(s)	constant (*)
		COLOR_HERITATE
		COLOR_OUTSIDE
		COLOR_INSIDE
		COLOR_INVERT
11	glazing bead type	constante (*)
		BEAD_NORMAL
		BEAD_EQUAL
		BEAD_SAFE
		BEAD_EQUALSAFE
		BEAD_ROUND
		BEAD_EQUALROUND
12	junction	constante (*)
		JUNCTION_MITRE
		JUNCTION_CONTINUE
		JUNCTION_STOP
13	sealing	constant (*)
		SEALING_GASKET
		SEALING_SILICONE
		SEALING_OTHER
14	route	constant (*)
		ROUTE_ATTACH
		ROUTE_DETACH

Parameters for Junction		
1	DLG_JUNCTION	constant
2	start object	object
3	show dialogue	boolean
4	Find	string
5	find a specific system	boolean
6	junction set	Integer (0..4)
	JUNCTION_SET1 (**)	
	JUNCTION_SET2 (**)	
	JUNCTION_SET3 (**)	
	JUNCTION_SET4 (**)	
	JUNCTION_SET5 (**)	
7	Except department(s) (**)	
8	Specifiek department(s) (**)	
9	Frame junction	boolean (*)
10	Frame connection	boolean (*)
11	Frame view	boolean (*)
12	Frame side	boolean (*)
13	Vent junction	boolean (*)
14	Vent connection	boolean (*)
15	Vent view	boolean (*)
16	Vent side	boolean (*)

```
REPLACEFILLING(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace filling

- sOld is the old filling code
- sNew is the new filling code
- bVerbose error messages (default not)

```
REPLACEFILLINGVARIETY(sOld, sNew[, bVerbose]
```
JoPPS 3.27

Find and replace variety for fillings

- sOld is the old variety code for fillings
- sNew is the new variety code for fillings
- bVerbose error messages (default not)

```
REPLACEFINISHING(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace finishing

- sOld is the old finishing code
- sNew is the new finishing code
- bVerbose error messages (default not)

```
REPLACEFINISHVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for finishing

- sOld is the old variety code for finishing
- sNew is the new variety code for finishing
- bVerbose error messages (default not)

```
REPLACEFRAMEVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for frames

- sOld is the old variety code for frames
- sNew is the new variety code for frames
- bVerbose error messages (default not)

```
REPLACEGLAZINGBEAD(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace glazingbead

- sOld is the old glazingbead code
- sNew is the new glazingbead code
- bVerbose error messages (default not)

```
REPLACEGLAZINGBEADVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for glazingbead

- sOld is the old variety code for glazingbead
- sNew is the new variety code for glazingbead
- bVerbose error messages (default not)

```
REPLACEINFORCEMENT(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace inforcement

- sOld is the old inforcement code
- sNew is the new inforcement code
- bVerbose error messages (default not)

```
REPLACEINFORCEMENTVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for inforcement

- sOld is the old variety code for inforcement
- sNew is the new variety code for inforcement
- bVerbose error messages (default not)

```
REPLACEPROFILE(sOld1, sOld2, sNew1, sNew2[, bVerbose])
```
JoPPS 3.27

Find and replace profile

- sOld is the old profile code
- sNew is the new profile code
- bVerbose error messages (default not)

```
REPLACEPROFILEVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for profile

- sOld is the old variety code for profile
- sNew is the new variety code for profile
- bVerbose error messages (default not)

```
REPLACESYSTEM(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace system

- sOld is the old system code
- sNew is the new system code
- bVerbose error messages (default not)

```
REPLACEVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety

- sOld is the old variety code for variety
- sNew is the new variety code for variety
- bVerbose error messages (default not)

```
REPLACEVENTVARIETY(sOld, sNew[, bVerbose])
```
JoPPS 3.27

Find and replace variety for vents

- sOld is the old variety code for vents
- sNew is the new variety code for vents
- bVerbose error messages (default not)

```
ReplaceStr( ? ) : ?
```
JoPPS/JCALL/JScripter – V2.82

To be documented.

`ReplaceText (Ssource,SsearchFor,SreplaceBy[,BignoreCase]) : S`

JoPPS/JCALL/JScripter

Replaces occurrences of one substring with another substring.

➘ Replace occurrences of a substring, specified by SsearchFor, with another substring, specified by SreplaceBy in a given string Ssource. If the optional argument Bignore is `TRUE` the comparison operation is case insensitive.

➚ The result of the replace operation.

StrReplace

`ReportHasMacro ([Dslot]) : B`

JoPPS

Checks if a report slot has an associated macro defined.

➘ Dslot is the optional slot index, if not specified the current slot is assumed.

➚ Returns `TRUE` if the specified slot has a macro.

Its better to specify a slot index as it is possible there is no current slot.

`ReportHasResult ([Dslot]) : B`

JoPPS

Checks if a report has a valid result.

➘ Dslot is an optional slot index, if not specified the current slot is assumed.

➚ Returns `TRUE` if the slot is up-to-date, `FALSE` if it is not.

Its better to specify a slot index as it is possible there is no current slot

`ReportInViewer () : Dslot`

JoPPS

Returns the slot index of the report currently being viewed.

➚ The slot index of the report slot being viewed in the result viewer, returns -1 if not result report is currently selected in the viewer.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

`ReportRan ([Dslot]) : B`

JoPPS

Returns if the specified report slot was included in the last report-phase. Likely the report slot generated a valid result.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.
ReportHasResult, UpdateReports

Its better to specify a slot index as it is possible there is no current slot.

`ResetReports( ? ) : ?`

JoPPS – V2.82

To be documented.

```
ResultsOk () : nResult
```

JoPPS 3.38 P5

Function to consider the results of a modified project as valid anyway

nResult := ResultOkay();

➚ nResult = 0 calculation was older than project
➚ nResult = 1 calculation was more recent than project

Calculate, SetBatchParams, SetPTableParams

```
ResultsValid () : B
```

JoPPS

Returns if whether the results in the result database are up-to-date.
(e.g. the "lightening" speedbutton in J*o*PPS is yellow)

The results are valid since the last calculation and till changes are made to a project or settings invalidating the current results in the result database.

➚ The state of the results in the result database.

Calculate, SetBatchParams, SetPTableParams

```
ROUND (D) : D
```

JoPPS/JCALL/JScripter

Rounds the argument to the nearest whole number.

The numeric value to be rounded.

➚ The rounded value.

TRUNC

```
RP(Sprog[,Sparams][,Bwait]) : Dexitcode
```

JoPPS/JCALL/JScripter – V2.82

This function is an alias for RunProgram.

RunProgram

```
RTrim (S) : S
```

JoPPS/JCALL/JScripter

Deletes trailing spaces and control characters from the given string.

➘ The string to trim.

➚ The "trimmed" string.

LTrim, Trim

RunMachine(Dslot[,Dmode][,Sfile]) : D

JoPPS - V2.83

Start generation format for the selected machine according to the specified mode.

➘ The Dslot argument specifies the slot of the machine to be executed (zero based).
➘ The optional Dmode argument overrides the current user interface mode. Possible values are:

Interface mode constants	Meaning
`OWMODE_POPUP`	show settings dialog and activate manually
`OWMODE_RUN`	don't show dialog and start automatically

➘ The optional Sfile argument overrides the default name for the output file.

➚ The returned value is the status, 0 means completed without errors otherwise the last errorcode is returned.

🖮 Example :

```
IF !ResultsValid() THEN
{
  Calculate();
  RunMachine(0,OWMODE_RUN); //start first machine defined without dialog
};
```

🕮 `SetupMachine`

RunProgram (Sprog[,Sparams][,Bwait]) : Dexitcode

JoPPS/JCALL/JScripter

Runs an external program. (.exe or .com)

➘ Sprog is the fully qualified filename of the program to run.
➘ Sparams is an optional string representing the commandline parameters passed to the program.
➘ Bwait is an optional boolean flag, passing a value of TRUE will cause the script execution to pause till the spawned program terminates.

➚ Dexitcode is the exitcode returned by the spawned program (if Bwait is `TRUE`), -1 indicates a failure finding or starting the program Sprog.

RunReport (Dslot[,Drunhow]) : B

JoPPS

Runs a J*o*PPS-Script result report.

➘ Dslot specifies the JoPPS-Script report slot. If Dslot does not point to a JoPPS-Script type report the function fails.
➘ The optional Drunhow argument specifies how the script is to be started by the interpreter.

Runhow constants	Meaning
`RUN_EDIT`	Opens the J*o*PPS-Script editor, does not execute the script.
`RUN_RUN`	Starts running the script immediately without opening the J*o*PPS-Script editor. The editor pops up when an error is encoutered.
`RUN_STEP`	Opens the J*o*PPS-Script editor and position the editor at the first statement.

➚ Returns `TRUE` if the report was executed without errors.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.
RunReportMacro

RunReportMacro ([Dslot]) : B

JoPPS

Runs a result report macro.

➘ Dslot is an optional slot index, if not specified the current slot is assumed.

➚ Returns `TRUE` if the macro executed without errors.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.
RunReport

Its better to specify a slot index as it is possible there is no current slot.

`RunScript (Sfn) : SDI`
JoPPS

Runs an external J*o*PPS-Script. Sfn specifies the J*o*PPS-Script source file.

➘ Sfn is the filename of the J*o*PPS-Script source file.
If no extensions is specified. JSS is assumed. (Since v2.06)
If no path is given the default JSS folder is assumed. (Since v2.06)

➚ The value returned is the result of the last statement executed in the script.

Make sure the file Sfn exists before passing it as an argument to this function.

`ScanLicense(): D`
JoPPS 3.30

Returns the status of the dongle, possible values are:

➚ DONGLE_NOTFOUND	no dongle hardware detected
➚ DONGLE_FOUND	dongle hardware was detected
➚ DONGLE_LICFOUND	corresponding license file found but not verified
➚ DONGLE_VERIFIED	license file found, license file verified ok
➚ DONGLE_TIMEOUT	license file found, license file timeout message
➚ DONGLE_EXPIRED	license file found, license file expired
➚ DONGLE_INVALID	license file found, dongle type invalid
➚ DONGLE_MAXUSERS	license file found, maximum users allowed
➚ DONGLE_MAXSLOTS	license file found, maximum stations allowed
➚ DONGLE_NOTLEGAL	dongle emulator detected, broadcast license info the value returned is the result of the last statement executed in the script.

`ScriptName ( ? ) : ?`
JoPPS/JCALL/JScripter – V2.82

To be documented.

`SearchFile( ? ) : ?`
JoPPS – V2.82

To be documented.

`Seek (Dh,Dpos) : D`
JoPPS/JCALL/JScripter

Positions the filepointer of a file.

➘ Dh is the filehandle of the file,
➘ Dpos is the new position of the filepointer. (in bytes from the beginning of the file)

➚ Returns the new filepointer position.

FilePos, FileLength

SelectDatabase ([Sdbid]) : B

JoPPS

Opens another or closes the current database.

➘ The optional Sdbid argument is the id of the database to open.
Not specifying a Passing an empty string logs out of the current database.

➚ Returns `TRUE` if successful.

GetDatabaseId, GetDatabaseDesc

SelectEditorDisplayMode(?) : ?

JoPPS – V2.82

To be documented.

SelectEditorFunction (Dfunid)

JoPPS - v2.70

Selects a specific editorfunction. The function must be available before it can be selected in the editor.

➘ Dfunid denotes the function to be selected.

funid	*editorfunction*
-21000	*Add opening*
-21001	*Remove atom*
-21002	*Redefine opening*
-21003	*Remove handle*
-21004	*Remove sill*
-21020	*Remove extra profile*
-21005	*Remove finishing*
-21006	*Remove accessories*
-21007	*Remove enforcement*
-21008	*Remove filling*
-21009	*Remove ventilator*
-21419	*Remove ventilator (on a T-mullion)*
-21010	*Remove georgian crosses*
-21011	*Remove glazing bead*
-21012	*Remove profile*
-21019	*Remove operation*
-21013	*Add vent*
-21021	*Add vent (2)*
-21014	*Remove framepart*
-21015	*Remove element*
-21016	*Remove filling*
-21017	*Remove ventpart*
-21018	*Remove vent*
-21100	*Add element*
-21101	*Add frameelement*
-21102	*Add ventelement*
-21103	*Add segment*
-21104	*Add T-mullion*
-21122	*Add T-mullion (framelevel)*
-21123	*Add T-mullion (ventlevel)*
-21106	*Add horizontal T-mullion*
-21107	*Add vertical T-mullion*
-21108	*Add fictive*
-21109	*Add general*
-21110	*Add internal*
-21111	*Add ventpart*
-21112	*Redefine ventpart*
-21120	*Add framepart*
-21126	*Add framepart (2)*
-21121	*Redefine framepart*
-21113	*Add splitter*
-21114	*Add origin*
-21115	*Add closure*
-21116	*Add broker*
-21117	*Add relative*

-21118	*Set X reference*
-21119	*Set Y reference*
-21203	*Add handle*
-21204	*Add sill*
-21214	*Add extra profile*
-21205	*Add finishing*
-21206	*Add accessories*
-21207	*Add enforcement*
-21208	*Add filling*
-21209	*Add ventilator*
-21418	*Add ventilator (on T-mullion)*
-21210	*Add georgian crosses*
-21211	*Add glazing bead*
-21212	*Add profile*
-21213	*Add operation*
-21300	*Move T-mullions*
-21301	*Align T-mullions*
-21302	*Align to reference*
-21303	*Align T-mullion in corner*
-21304	*Reshape framecorner*
-21305	*AdjustStijlen*
-21306	*Remove T-mullions*
-21307	*Align vertical*
-21308	*Align horizontal*
-21309	*Align equal*
-21310	*Edit node*
-21311	*Merge*
-21312	*Split*
-21313	*Insert splitter*
-21314	*Delete splitter*
-21400	*Add vertical*
-21401	*Add horizontal*
-21402	*Exchange colours*
-21403	*Static Ix*
-21404	*Static Iy*
-21448	*Select section*
-21405	*Add section*
-21406	*Remove section*
-21449	*Print sections*
-21407	*List sections*
-21411	*Set section*
-21408	*Swap ratio*
-21409	*Offset*
-21447	*Mirror*
-21410	*Calculate weight*
-21412	*Make vent*
-21413	*Add plint*
-21414	*General selection*
-21415	*Change measurements*
-21124	*Store frame*
-21125	*Store vent*
-21420	*Select profielen*
-21421	*Select versterkingen*
-21422	*Select vullingen*
-21423	*Select georgian crosses*
-21424	*Select finishes*
-21425	*Select glazing bead*
-21426	*Select ventilator*
-21417	*Select ventilator (on T-mullion)*
-21427	*Select functional accessories*
-21428	*Select extra accessories*
-21429	*Select operations*
-21430	*Select model*
-21431	*Select sill*
-21432	*Select ventpart*
-21433	*Select vent*
-21446	*Select extra profiles*

SelectEditorMode(?) : ?

JoPPS – V2.82

To be documented.

```
SendMail( ? ) : ?
```

JoPPS/JCALL/JScripter – V2.82

To be documented.

```
SetActiveProjectIndex (D) : Dindex
SetActiveProjectIndex (S) : Dindex
```

JoPPS

Make a specific project in the projectpool the active project.

➘ D-type is project index in the projectpool,
➘ S-type is the name of project to select.

➚ Returns the index of the active project.

```
SetAutoSaveTimer(Bstate) : Bstate
```

JoPPS 3.28

Set the “Autosave according to interval of time’ setting.

True	activated
False	deactivated

```
SetBatchParams (Sbatchcode[,Dno[,Dcarrier[,Dcabins]]])
```

JoPPS

Sets initial batch parameters prior starting batch calculations.

➘ Sbatchcode is the batch reference to be used,
➘ Dno is the initial value for the frame counter used to number the frames processed in the run (default=1),
➘ Dcarrier sets the starting carrier number (default=1)
➘ Dcabins is the number of cabins for one carrier (default=16).

If one of the arguments is not specified its previous value remains in effect,
If no arguments are passed the batch parameters are reset to their defaults
(batchcode = name of the first project in the batch, Dno=1, Dcarrier=1, Dcabins=16).

Example :

```
mode := GetCalcMode();
IF mode <> CALCMODE_BATCH THEN SetCalcMode(CALCMODE_BATCH);
SetUI(ProjectCount()>1);
batch := Upper(ChangeFileExt(ExtractFilename(GetProjectFilename(0)),''));
SetBatchParams(batch,1,1,16);
Calculate();
SetCalcMode(mode);
```

Calculate, SetCalcMode, SetUI, SetPTableParams

Passing an empty string for the Sbatchcode will cause the name of the first project in the batch to be used.

```
SetBit() :
```

JoPPS

To be documented.

`SetCalcBehaviour (Dcalcbehaviour)`

JoPPS

Sets the calculation behaviour.

➘

Calculation behaviour constants	Meaning
`CALCBEHAVIOUR_USER`	?
`CALCBEHAVIOUR_DEFAULT`	Calculate
`CALCBEHAVIOUR_NOREPORTS`	Calculate (without reports)
`CALCBEHAVIOUR_OFFER`	Calculate (commercial)
`CALCBEHAVIOUR_ORDER`	Calculate (order)
`CALCBEHAVIOUR_PRODUCTION`	Calculate (production)

GetCalcMode

`SetCalcMode (Dcalcmode)`

JoPPS

Sets the calculation mode.

Refer to the explanation of the `Calculate` function for a discussion of the calculation mode constants.

🕮 Refer to *3. Using JoPPS-Script in JoPPS* for more information on the calculation mode.
GetCalcMode, SetUI, SetBatchParams, SetPTableParams

`SetCalcPerBatchType( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetCaption (S)`

JCALL

Sets the caption (e.g. the text of the window title bar) of the JCALL console window.

➘ S is the caption.

`SetCurDir (Sdir)`

JoPPS/JCALL/JScripter

Changes the current directory. Sdir can be a complete folder spec. including a drive identifier.

➘ Sdir is the new current directory.

GetCurDir

`SetCurrentLanguage( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetDebug( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetEnableActions (Denable)`

JoPPS - v2.0

Controls the state of the EnableActions flag. Can be used to disable the generation of instructions to interface machining centers. Is only of use for licenses having the MC option. Normally this flag is set manually by the user.

➘ Denable is the new state for the EnableActions flag.

GetCurDir

`SetFileDate (Dh, Ddatetime) : B`

JoPPS/JCALL/JScripter

Sets the date-and-time stamp of the specified filehandle.

➘ Dh is the filehandle,
➘ Ddatetime is the date-time value.

➚ Returns `TRUE` is successful, `FALSE` if not.

GetFileDate, FileAge, DateToStr, DateTimeToStr, FormatDateTime

`SetGUIKind( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetMainTab (Dtabindex)`

JoPPS

Sets the J*o*PPS maintab.

➘ Dtabindex is the index of the tab to set.

Maintab constants	Meaning
`TAB_PROJECT`	Selects the project tab
`TAB_EDITOR`	Selects the editor tab
`TAB_RESULTS`	Selects the result tab

`SetMsgPaneMode( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetParam (Sparam[,Svalue[,Dvalue]])`

JoPPS

Sets a J*o*PPS parameter to the specified value.

➘ Sparam is the name of the parameter (not between % characters !),
➘ Svalue is the new value for the parameter.
If no Svalue argument is passed the parameter is assigned an empty string.
➘ Dvalue specifies parameter level (PL_SYSTEM, PL_REPORT, ... , PL_USER)
If no Dvalue argument is passed the parameter level is PL_REPORT

GetParam

`SetPRIORPTPath( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetPTableParams ([Btxtfmt[,Bstdfmt[,Btrffmt]]])`

JoPPS

Sets pricetable parameters prior starting calculations.

If Btxtfmt is `TRUE` a DOS ascii textfile is generated,
If Bstdfmt is `TRUE` then a standard price output file will be generated,
If Btrffmt is `TRUE` a price tariff output file will be generated.

Calculate, SetBatchParams, SetUI

`SetResultParam (DparamId,DSI) : DSI`

JoPPS

Sets the value of the specified result parameter.

➘ The id of the parameter to set,
➘ The second argument (D-type, S-type or I-type) is the new value for the parameter specified.

➚ Returns the previous value of the specified parameter.

GetResultParam

Future implementations will allow you to set global and/or report specific parameters.

`SetSaveToDisk (Benable)`

JoPPS

Sets the state of the "SaveToDisk" flag. The SaveToDisk flag controls whether or not results are written to disk.

➚ Benable is the new state of the SaveToDisk flag.

`SetSendToProd( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetSyntax( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetUI (Benable)`

JoPPS

Sets the internal UI flag. The UI flag indicates whether or not J*o*PPS should popup the batch-, pricetableparameter or newproject dialog. The state of the UI flag is ignored when working with J*o*PPS interactively.

Benable is the new state of the UI flag, should be `TRUE` or `FALSE`.

GetUI

`SetupMachine(Dslot) : D`

JoPPS – V2.83

Start generation format for the selected machine according to the specified mode.

➚ The Dslot argument specifies the slot of the machine to be parameterised (zero based).
➚ The returned value is the status, 0 means completed without errors otherwise the last errorcode is returned.

⌨ Example :
```
SetupMachine(0); //calls setup dialog first defined machine in JoPPS
```

🕮 `RunMachine`

`SetUseScrapMan( ? ) : ?`

JoPPS – V2.82

To be documented.

`SetWallPaper (Sfn) : B`

JoPPS - v2.0

Specifies a new file to be used as wallpaper. (the background displayed when no projects are loaded)

➘ Sfn is the name of the file. The file can be a windows bitmap (.BMP) or an HTML document (.HTM)

SetWallPaperSource

`SetWallPaperSource (Sfn) : B`

JoPPS - v2.0

Specifies the html source to be used as the new wallpaper.

Shtml is a string holding an entire HTML document. You can pass the result of the StringToHTML function to render a text on the wallpaper.

StringToHTML, SetWallPaper

`ShowDatabaseSelection ()`

JoPPS

Opens the J*o*PPS database selection dialog. Execution continues after the user closed the database selection dialog either by selecting a new database or by just closing the dialog.

ShowDataDlg (Ddlg) : B

JoPPS

Opens the specified J*o*PPS data dialog.

➘ The Ddlg argument identifies the data dialog to show.

➚ Returns `TRUE` if the dialog was closed pressing the OK button, `FALSE` if not.

Dialog constants	Identifies the
`DLG_CLIENT`	Client dialog
`DLG_FINISH`	Finish dialog
`DLG_SYSTEM`	System dialog
`DLG_PRODUCT`	Profile product dialog
`DLG_PROFILE`	Profile parameters dialog
`DLG_COMBINATION`	Profile combination dialog
`DLG_GLAZINGBEAD`	Profile glazing bead dialog
`DLG_REINFORCEMENT`	Profile reinforcement dialog
`DLG_ACCESSORY`	Accessories dialog
`DLG_ACCSET`	Accessories table dialog
`DLG_FILLING`	Filling dialog
`DLG_FINISHES`	Finishes dialog
`DLG_PRICE`	Price parameters dialog
`DLG_PRICESTANDARD`	Price standard dialog
`DLG_PRICETARIFF`	Price tariff dialog
`DLG_PRICEBLOCK`	Price block dialog
`DLG_NORM`	Norm (wind) dialog
`DLG_SET`	Accessories set dialog
`DLG_ACTION`	Action dialog
`DLG_OPERATION`	Machining operations dialog
`DLG_TASK`	Task dialog
`DLG_FRAME`	Frame library dialog
`DLG_VENT`	Vent library dialog

ShowJieViewer ([Sjiefn])

JoPPS - v2.0

Opens the JIE fileviewer and shows the contents of the JIE file specified. Execution continues after the user closed the fileviewers window.

➘ Sjiefn is the optional fn of the jie file to examine. If no name is specified the user is prompted to pick a valid jiefile.

ShowMessage (SD)

JoPPS/JCALL/JScripter

Displays a message in a message dialog. The title is the name of the script being executed. The message is the concatenation of the arguments passed.

One or multiple S or D-type arguments.

MsgErr, MsgBox

ShowProdView(?) : ?

JoPPS – V2.82

To be documented.

ShowProjectManager ()

JoPPS

Opens the J*o*PPS project manager. Execution continues after the user closed the project manager window.

```
ShowResult ([Dslot])
```

JoPPS

Views the result of a given report slot in the result viewer window.

➘ Dslot is the report slot index of the result to view.

🕮 For an overview of slot index constants refer to the discussion of the `TagReport` function.

Its better to specify a slot index as it is possible there is no current slot.

```
ShowToDoList( ? ) : ?
```

JoPPS – V2.82

To be documented.

```
ShowThumbDlg() : Skey
```

JoPPS – V3.28

Client/Sypplier table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight[,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_CLIENT)
Scode	client/supplier code

Finish tabel:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_FINISH)
Scode	finish code

Profile product table:

ShowThumbDlg (Dtable,Scode,Dlen,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_PRODUCT)
Scode	profile product code
Dlen	profile product lengte

Profile properties table:

ShowThumbDlg (Dtable,Ssystem,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_PROFILE)
Ssystem	profile propertie system
Scode	profile propertie code

Accessories table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_ACCESSORY)
Scode	accessories code

Glazing table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_FILLING)
Scode	glazing code

Window finish table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_FINISHES)
Scode	window finish code

Task table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]]) : Skey

Dtable	table identifier (DLG_TASK)
Scode	task code

Manipulations table:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_OPERATION)
Scode	manipulation code

Models table:

ShowThumbDlg (Dtable,Scode,,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtable	table identifier (DLG_FRAME)
Scode	model code

Vents table:

ShowThumbDlg (Dtable,Scode,[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dtabletable identifier (DLG_VENT)
Scode vent code

Optional properties:

ShowThumbDlg (Dtable,Scode,[,Dkind[,Dmode[,Dsize[,Dwidth[,Dheight [,Bhint[,Dfilter]]]]]]]) : Skey

Dkindthumbnail kind		
	USAGE_DRW	frontview (standard)
	USAGE_CAD	section
Dmodethumbnail layout		
	THUMB_GRID	grid
	THUMB_GRID	list
Dsize	thumbnail size	
Dwidththumbnail # horizontal		
Dheightthumbnail # vertikal		
Bhint	information hints	
	True	show
	False	hide
Dfiltercategorie filter		

➚ Skey

Example Thumbs Dialogs*:*

```
/* ------------------------------------------------------------ -- */
/* Example Thumbs Dialogs.jss                                       */                         /*
Example for selection of DB record in scripting.                 */          */
/* TechWIN Software BVBA 2014 (c) - Gunter Selleslagh */
/* --------------------------------------------------------------- */
/* create filling object */
contact := CONTACTS.Create();
profile := PRODUCTS.Create();
fillings := FILLING.Create();
/* show contactss dialog */
sContact := 'DC';
sKey := ShowThumbDlg(DLG_CLIENT,sContact,USAGE_CAD,THUMB_GRID,50);
if sKey <> '' then { ShowMessage(sKey); };
/* show colors dialog */
sColor := '3511';
sKey := ShowThumbDlg(DLG_FINISH,sColor,USAGE_CAD,THUMB_GRID,60,0);
if sKey <> '' then { ShowMessage(sKey); };
/* show profiles dialog */
sProfile := '5001';
aLength := 6000.0;
sKey := ShowThumbDlg(DLG_PRODUCT,sProfile,aLength,USAGE_CAD,THUMB_GRID,75,1);
if sKey <> '' then { ShowMessage(sKey); };
/* show fillings dialog */
sFilling := '1-PANEEL-WIT';
sKey := ShowThumbDlg(DLG_FILLING,sFilling,USAGE_DRW,THUMB_GRID,100);
if sKey <> '' then { ShowMessage(sKey); };
/* free objects */
contact.Free();
profile.Free();
fillings.Free();
```

`ShowWizard ()`

JoPPS

Opens the J*o*PPS wizard dialog.

`SIN (D) : D`

JoPPS/JCALL/JScripter

Sin returns the sine of an angle.

➘ Dangle is the angle in degrees.

➚ The sine of the given angle.

ASIN, COS, ACOS, TAN, ATAN

`SM(SD)`

JoPPS/JCALL/JScripter – V2.82

This function is an alias for ShowMessage.

ShowMessage

`SQRT (D) : D`

JoPPS/JCALL/JScripter

Sqrt returns the square root of the argument.

D-type value.

➚ Square root the argument.

`START(S) : I`

JoPPS/JCALL/JScripter

Instanciates an OLE automation server and returns a variable holding its IDispatch interface.
When successfull the returned variable can be used as an (ActiveX) object to access the
server's methods and properties.

➘ S specifies the name of the OLE automation server to initiate, eg. `"word.application"` will start MS Word and return an IDispatch interface to the word application.

➚ Returns a variable (I) holding the IDispatch interface to the automation objects properties and methods. Returns `FALSE` when failed.

⌨ The following example creates a new document in MS Word:

```
word := START("word.application");
word.Visible := TRUE;
word.Documents.Add();
```

💣 Before using the returned variable make sure it is a a valid IDispatch object. Use the `IsIDispatch` function for this purpose.

`StrAlignL (Ssource,Dmaxlen [,Spad]) : S`

JoPPS/JCALL/JScripter

Returns a left aligned string with a given maximum length.

➘ Ssource is the string to align,
➘ Dmaxlen is the maximum length of the result string,
➘ The optional Spad argument is used to fill the string up to the given maximum length (default = blank).
If the length of the source string exceeds the given maximum length the string is not truncated.

➚ The aligned string.

StrAlignR

```
StrAlignR (Ssource,Dmaxlen [,Spad]) : S
```
JoPPS/JCALL/JScripter

Returns a right aligned string with a given maximum length.

➘ Ssource is the string to align,
➘ Dmaxlen is the maximum length of the result string,
➘ The optional Spad argument is used to fill the string up to the given maximum length (default = blank).
If the length of the source string exceeds the given maximum length the string is not truncated.

➚ The aligned string.

StrAlignL

```
StrDelete (Ssource, Dstart [,Dcount]) : S
```
JoPPS/JCALL/JScripter

Deletes a number of characters from a string.

➘ Ssource is the source string,
➘ Dstart is the position in the source string where deleting should start,
➘ The optional Dcount parameter tells how many characters should be deleted, if not specified
all characters are deleted up to the end of the string

➚ The resulting string.

StrDeleteL, StrDeleteR

```
StrDeleteL (Ssource, Dcount) : S
```
JoPPS/JCALL/JScripter

Deletes a number of leading characters from a string.

➘ Ssource is the source string,
➘ Dcount is the number of characters to delete.

➚ The resulting string.

StrDelete, StrDeleteR, TrimL

```
StrDeleteR (Ssource, Dcount) : S
```
JoPPS/JCALL/JScripter

Deletes a number of trailing characters from a string.

➘ Ssource is the source string,
➘ Dcount is the number of characters to delete.

➚ The resulting string.

StrDelete, StrDeleteL, TrimR

```
StringToColor( ? ) : ?
```
JoPPS/JCALL/JScripter – V2.82

To be documented.

```
StringToHTML (S) : Shtml
```
JoPPS/JCALL/JScripter -v2.5

Converts a string into a valid HMTL document.

➘ S is the contents of the HTML document to create.

➚ The HTML source.

`StrInList (Ssub,Sstr[,Sdelimiter]) : B`

JoPPS/JCALL/JScripter -v2.70

Returns True when a given substring appears in a given string holding a list of values.

➘ Ssub is the substring to search for, Sstr is the string holding a comma separated list of all possible values,
➘ Sdelimiter can be used to specify an alternative separator.

➚ True or False.

StrPos

`StrLeft (Ssource, Dcount) : S`

JoPPS/JCALL/JScripter

Returns a left portion of a given string.

➘ Ssource is the source string,
➘ Dcount is the number of characters to copy to the result string.

➚ The resulting string.

StrRight

`StrLen (S) : D`

JoPPS/JCALL/JScripter

Returns the actual length of the given string.

➘ S is the input string.

➚ The length of the string.

StrSize

`StrPos (Ssub, Ssource) : D`

JoPPS/JCALL/JScripter

Returns the position of a substring within a given string.

➘ Ssub is the string to search for,
➘ Ssource is the source string.

➚ Returns the index of the substring in the source string. If the substring is not found in the source string a value of 0 is returned.

`StrReplace (Ssource, Ssearch, Sreplace) : S`

JoPPS/JCALL/JScripter

Replaces or deletes characters of a string.

➘ Ssource is the source string,
➘ Ssearch is a string holding all characters to delete or to replace from the Ssource string,
➘ Sreplace is a string holding the replacement characters for each Ssearch string in the Ssource string.

If Sreplace is shorter than Ssearch, characters from Ssearch having no corresponding character in Sreplace will be deleted. No wildcards can be used.

➚ The resulting string.

ReplaceText

StrRight (S, Dcount) : S

JoPPS/JCALL/JScripter

Selects a right portion of a given string.

➘ Ssource is the source string,
➘ Dcount is the number of characters to copy to the result string.

➚ The resulting string.

StrLeft

StrSize (S) : Dsize

JoPPS/JCALL/JScripter

Returns the allocated size of a given string.

➘ S is the source string.

➚ Dsize is the allocated size of the given string, not its length.

StrLen

StrToDate (Sdate) : Ddate

JoPPS/JCALL/JScripter - v2.20

Converts a string representing a date into a D-type date.

➘ Sdate is the date represented as a S-type value, it must be correctly formated (D/M/Y).

➚ Returns the date as a D-type value.

DateToStr, DateTimeToStr

StrToFile (Stext,Sfn) : B

JoPPS/JCALL/JScripter

Writes a string to a textfile.

➘ Stext is the string,
➘ Sfn is the file of the textfile to create.

➚ Returns `TRUE` is successful.

FileToStr

StrToNum (S) : D

JoPPS/JCALL/JScripter

Converts a string argument to a numeric value.

➘ S is a string representing a numeric value.

➚ Returns the result of the type conversion, a numeric (D-type) value.

If the string S does not contain a valid numeric value, (numbers, point, minus sign) a run-time error is generated and the script execution is halted.

NumToStr, IntToStr, IntToStr0

StrToNum (S, Ddefault) : D

JoPPS/JCALL/JScripter

Converts a string argument to a numeric value, if the string S does not contain a valid numeric value the default value is returned.

➘ S is a string representing a numeric value.
➘ Ddefault is the default numeric value to be returned when the conversion of S fails.

➚ Returns the result of the type conversion, a numeric (D-type) value.

NumToStr, IntToStr, IntToStr0

StrToTime (Stime) : Dtime

JoPPS/JCALL/JScripter - v2.20

Converts a string representing a time into a D-type time.

➘ Stime is the time represented as a S-type value, it must be correctly formated (H: M:S).

➚ Returns the time as a D-type value.

TimeToStr, DateTimeToStr

SubmitOrder (Beos)

JoPPS/JCALL/JScripter 3.32 P2

Send results to the order management module

➘ Beos can be False or True, submit or not

Example :

```
/* XML genereren      */
/* Gunter Selleslagh - april 2016*/

/* Current project */
CurPro := GetCurrentProject();
if CurPro = Nil then halt;

/* Calculate project */
Calculate(CALCMODE_PROJECT,XML_NONE);

/* Submit order */
SubmitOrder(True);
```

SubStr (Ssource, Dstart [,Dcount]) : S

JoPPS/JCALL/JScripter

Returns a substring from a given string.

➘ Ssource is the source string,
➘ Dstart is the start position of the substring,
➘ Dcount is the length of the substring, thus the number of characters to copy.
If Dstart + Dcount exceeds the length of the source string all remaining characters of the source string are copied.

➚ The substring.

StrPos, StrLeft, StrRight

SubStrCnt (Ssource[,Sdelimiter]) : D

JoPPS/JCALL/JScripter

Returns the number of substrings in a "collection" string.

A collection string is a string consisting of substrings separated by delimiter characters.
The delimiter character, if not specified, is the comma ','.

- ➘ Ssource is the collection string,
- ➘ The optional Sdelimiter argument is the delimiter.

- ➚ The number of substrings in the collection string.

`GetSubStr`

`TagReport (Dslot,B)`

JoPPS

Tags or untags a report slot.

➘ Dslot is the slot index,
➘ B denotes if the specified slot should be tagged (TRUE) or untagged (FALSE).

Slot index constants	Meaning	Type	Value
`SLOT_ALL`	Represents all report slots		-1
`SLOT_OUTLINES`	Outlines	HTML	0
`SLOT_BILL_OF_MATERIAL`	Bill of material	HTML	1
`SLOT_CUTTING_LIST`	Cutting list	HTML	2
`SLOT_ACCESSORIES_LIST`	Accessories list	HTML	3
`SLOT_FILLING_LIST`	Filling list	HTML	5
`SLOT_FINISHING_LIST`	Finishing list	HTML	6
`SLOT_FRAME_PER_PAGE`	Frame per page	HTML	7
`SLOT_ASSEMBLY_PER_PAGE`	Assembly per page	HTML	8
`SLOT_ORDERLIST_PER_SUPPLIER`	Orderlist per supplier	HTML	9
`SLOT_ORDERLIST_PER_PRODUCT`	Orderlist per product	HTML	10
`SLOT_OPTIMIZATION`	Optimization	HTML	11
`SLOT_ESTIMATION_DETAIL`	Estimation detail	HTML	12
`SLOT_ESTIMATION_SUMMARY`	Estimation Summary	HTML	13
`SLOT_CALCULATION_DETAIL`	Calculation Detail	HTML	14
`SLOT_CALCULATION_SUMMARY`	Calculation Summary	HTML	15
`SLOT_SECTIONS_HTML`	Sections	HTML	16
`SLOT_OFFER_HTML`	Offer	HTML	17
`SLOT_OFFER_OUTLINES`	Offer outlines	HTML	18
`SLOT_INVOICE`	Invoice	HTML	19
`SLOT_CONFIRMATION`	Confirmation note	HTML	20
`SLOT_DELIVERY_NOTE`	Delivery note	HTML	21
`SLOT_STATISTICS`	Statistics	HTML	22
`SLOT_OUTLINES_LBL`	Outlines	LABEL	23
`SLOT_FINISHING_LBL`	Finishing	LABEL	24
`SLOT_OPTIMIZATION_LBL`	Optimization	LABEL	26
`SLOT_ORDERLIST_PER_SUPPLIER_TEXT`	Orderlist per supplier	TEXT	27
`SLOT_ORDERLIST_PER_PRODUCT_TEXT`	Orderlist per product	TEXT	28
`SLOT_OFFER_SCRIPT`	Offer	SCRIPT	29
`SLOT_ORDERLIST_PER_SUPPLIER_SCRIPT`	Orderlist per supplier	SCRIPT	30
`SLOT_ORDERLIST_PER_PRODUCT_SCRIPT`	Orderlist per product	SCRIPT	31
`SLOT_SECTIONS_SCRIPT`	Sections	SCRIPT	32
`SLOT_BILL_OF_MATERIAL_TEXT`	Bill of material	TEXT	33
`SLOT_CUTTING_LIST_TEXT`	Cutting list	TEXT	34
`SLOT_ACCESSORIES_LIST_TEXT`	<currently not used>	TEXT	35
`SLOT_FINISHING_LIST_TEXT`	<currently not used>	TEXT	36
`SLOT_FRAME_LBL`	<currently not used>	TEXT	37
`SLOT_USER1`	User defined report #1	HTML	38
`SLOT_USER2`	User defined report #2	HTML	39
`SLOT_USER3 (>2.0)`	User defined report #3	HTML	40
`SLOT_USER4 (>2.0)`	User defined report #4	HTML	41
`SLOT_USER5 (>2.0)`	User defined report #5	HTML	42
`SLOT_MC1_LBL (>2.10)`	Labels MC-1	LABEL	63
`SLOT_MC2_LBL (>2.10)`	Labels MC-2	LABEL	62

UpdateReports, Calculate, SetCalcMode, GetCalcMode

📖 Refer to *3. Using JoPPS-Script in JoPPS* for more information on report slots.

`TAN (Dangle) : D`

JoPPS/JCALL/JScripter

Returns the tangent of the D-type argument.

➘ Dangle is the angle in degrees.

➚ Returns the tangent of the given angle.

ATAN, SIN, ASIN, COS, ACOS

`Terminate () : B`

JoPPS – V2.82

Terminates JoPPS session. Programmer has responsibility of saving all changed projects and/or data before calling this function within a script.

No arguments

Returns `TRUE` if succesfull, `FALSE` if not.

`TextIn( ? ) : ?`

JCALL – V2.82

To be documented.

`TextOut (S)`

JoPPS/JCALL – V2.82

Writes a string to the standard output stream.

➘ S is the string to be written.

LineOut, CharOut

`TimeToStr ([Ddatetime]) : S`

JoPPS/JCALL/JScripter

Converts the time part of a given date-time value into a string.
(using the ShortTimeFormat setting from the Windows regional settings)

➘ Ddatetime is the date-time value to convert.

➚ The time string.

Now, DateToStr, DateTimeToStr, FormatDateTime

`Trim (Ssource) : S`

JoPPS/JCALL/JScripter

Trim deletes leading and/or trailing spaces and control characters.

➘ Ssource is the string to trim.

➚ The trimmed string.

LTrim, RTrim

`TRUNC (D) : Dint`

JoPPS/JCALL/JScripter

Truncates a number to an integer.

➘ D is the number to be truncated.

➚ The integer part of the number.

ROUND, CEIL, FLOOR

`UpdateDrawings ()`

JoPPS - v2.70

Redraws all the glyphs currently displayed in the GUI (eg. editor, assemblyselection, etc.)

`UpdateEditor ()`

JoPPS - v2.70

Redraws the editor

`UpdateReports () : B`

JoPPS

Updates the tagged report slots by rerunning the report-phase of the calculations.

➚ Returns `TRUE` if the report-phase started,
returns `FALSE` if the results in the result database are <u>not</u> up-to-date.

Calculate, ResultsValid

Use `Calculate` instead of `UpdateReports` if the results in the result database are no longer up-to-date.
Cannot be used when the calculation mode is `CALCMODE_PTABLE`.

`Upper (Ssource) : S`

JoPPS/JCALL/JScripter

Upper converts a string to uppercase.

➘ Ssource is the string to convert.

➚ The string in uppercase.

Lower

`ViewToBitmap (Oatom,Sname,Dwidth,Dheight,Dside`
`[,Dmeasure[,Dcolor[,Dscale[,Dscenario[,Dview[,Dresol]]]]]]) :B`

JoPPS/JCALL/JScripter

Create a bitmap of the left, right, top or bottom view of a group or part.

➘	*Oatom*	*group/part object*	
	Sname	*filename of the drawing*	
	Dwidth	*width drawing*	
	Dheight	*height drawing*	
	Dside	*side (left=1, right=2, below=3, above=4)*	
	Dmeasure	*measure?*	*(*)*
	Dcolor	*in color?*	*(*)*
	Dscale	*scale (=1.0)*	*(*)*
	Dscenario	*scenario*	*(*)*
	Dview	*view (according to group=-1, inside=0, outside=1)*	*(*)*
	Dresol	*resolution (=120.0)*	*(*)*

➚ Bitmap drawing.

Arguments marked with (*) are optional

`ViewToMetafile (Oatom,Sname,Dwidth,Dheight,Dside`
`[,Dmeasure[,Dcolor[,Dscale[,Dscenario[,Dview[,Dresol]]]]]]) :B`

JoPPS/JCALL/JScripter

Create a metafile of the left, right, top or bottom view of a group or part.

➘	*Oatom*	*group/part object*	
	Sname	*filename of the drawing*	
	Dwidth	*width drawing*	
	Dheight	*height drawing*	
	Dside	*side (left=1, right=2, below=3, above=4)*	
	Dmeasure	*measure?*	*(*)*
	Dcolor	*in color?*	*(*)*
	Dscale	*scale (=1.0)*	*(*)*
	Dscenario	*scenario*	*(*)*
	Dview	*view (according to group=-1, inside=0, outside=1)*	*(*)*
	Dresol	*resolution (=120.0)*	*(*)*

➚ Metafile drawing.

Arguments marked with (*) are optional

`Wait (Dmillisec)`

JoPPS/JCALL/JScripter - v2.0

Suspends script execution for Dmillisec milliseconds.

`WeekOfYear ([Ddatetime]) : Dweek`

JoPPS/JCALL/JScripter

Returns the week of the year.

➘ If no parameter is given todays date is used as input, the optional Ddatetime argument specifies an alternative date.

➚ Dweek is the week of the year.

Example :

```
OutputMsg("We are today in week "+IntToStr(WeekOfYear)));
```

Now, Day, Month, Year, DayOfWeek, DayOfYear

`WindowMaximize ()`

JoPPS/JCALL/JScripter

Maximizes the J*o*PPS main window.

`WindowRestore ()`

JoPPS/JCALL/JScripter

Restores the J*o*PPS main window to its original size.

`Write (Dh,Sbuf,Slen)`

JoPPS/JCALL/JScripter

Writes information to a file.

➘ The output file specified by the file handle Dh,
➘ Sbuf is a S-type variable holding the data to be written,
➘ Slen is the maximum number of bytes to write.

➚ The number of bytes actually written.

WriteLn, TextOut, CharOut

`WriteLn (Dh[,SD])`

JoPPS/JCALL/JScripter

Writes information to a textfile as a line of text.

➘ Dh is the filehandle followed by one or multiple S or D-type parameters.
The parameters are concatenated and make up one line of text in the textfile.

OpenFile, OpenWrite, Write, TextOut, CharOut

`WriteStr (Dh,SD)`

JoPPS/JCALL/JScripter

Writes information to a textfile.

➘ Dh is the filehandle followed by one or multiple S or D-type arguments.

OpenFile, OpenWrite, Write, TextOut, CharOut

`XmlExport([Dxml])`

JoPPS/JCALL/JScripter 3.32 P2

Generate XML after calculations

➘ Dxml is one of the following XML scenario’s

XML_GENERAL
XML_REQUEST
XML_SELLER
XML_ORDER
XML_RESERVE

Example :

```
/* XML genereren      */
/* Gunter Selleslagh - april 2016*/

/* Current project */
CurPro := GetCurrentProject();
if CurPro = Nil then halt;

/* Calculate project */
Calculate(CALCMODE_PROJECT,XML_NONE);

/* Generate XML */
XmlExport(XML_GENERAL);
```

`Year ([Ddatetime]) : Dyear`

JoPPS/JCALL/JScripter

Returns the year.

➘ If no parameter is given todays date is used as input, the optional Ddatetime argument specifies an alternative date.

➚ Dyear is the year.

Now, Day, Month, DayOfWeek, WeekOfYear, DayOfYear

`YN(SD[Breply]) : Breply`

JoPPS/JCALL – V2.82

This function is an alias for AskYN.

AskYN

10. Appendices

10.1. Errormessages

The following list presents most common run-time errors.

Error	Meaning
`1`	Unbalanced braces. Error in expression.
`2`	Nothing to do. Empty script source, no statements found
`3`	Invalid return type. The type returned is of an invalid type.
`4`	Unknown function. The interpreter encountered an unknown function.
`5`	Invalid argument. Co-processor floating point error caused by an invalid argument. (eg. the `SQRT` function is called with a negative argument)
`6`	Invalid operator. Co-processor invalid operand.
`7`	Overflow. An numeric overflow. Result of expression is too big.
`8`	Math underflow. A numeric underflow. Result of expression is too small.
`9`	Divide by zero. A division by 0 was attempted.
`10`	Incompatible binary operands. Operands cannot be used together with this operator. (eg. cannot add a D-type and S-type value.)
`11`	Incompatible unary operand. The operand is used in an improper way.
`12`	No expression. An expression was expected here.
`13`	Invalid character. An invalid character was encountered in the source.
`14`	Unknown variable. A unknown variable was encountered.
`15`	Unexpected end of double. Wrongly specified D-type value.
`16`	Unexpected end of string. S-type constant was not closed.
`17`	Cannot parse double. S-type to D-type conversion failed.
`18`	Double out of range. A D-type value exceeds the numeric limits.
`19`	Invalid unary operator. An unary operator (has 2 operands) is used in an improper way.
`20`	Invalid binary operator. An binary operator (has only 1 operand) is used in an improper way.
`21`	Invalid start of expression. An expression begins with an invalid character. (eg. expression begins with a binary operator or an opening brace `)1+1)`)
`22`	Invalid follow up in expression. The expression contains an error. (eg. `(1+1)(2+2)`)
`23`	Invalid end of expression. The expression contains an error. Probably the end of expression is missing. (eg. `(1+2`)
`24`	Invalid level in expression. The expression contains an error. Probably an error with braces.
`25`	Comma not allowed here. A comma is encountered where it is not expected.
`26`	An argument passed to the `CHR` function is not a valid character.
`27`	Bad variable syntax. (eg. `JOPPS..ProjectCount` instead of `JOPPS.ProjectCount`)
`28`	Invalid sub-expression. Brackets ([]) not in balance.
`30`	Server not started. Triggered by the `START` function when the requested OLE server is unknown.
`32`	Unknown server method. An referenced OLE server method is unknown to the server.
`33`	Unknown server property. An referenced OLE server property is unknown to the server.
`34`	Only one statement allowed. The interpreter encountered a ; where it is not expected.
`35`	Statement expected. The interpreter expects at least one statement within the statement blocks of an `IF-THEN-ELSE` or `WHILE-DO` construction.
`36`	Unexpected end of script. The end of the source is encountered but not yet expected.
`37`	Semicolon expected. The interpreter expects a ';' here but finds something else. (eg. the interpreter expects a ; after a label in a `GOTO` statement, after a `CONTINUE` or

	BREAK statement)
38	Invalid use of reserved word. A reserved word is encountered but not expected.
39	No corresponding loop statement. A BREAK or CONTINUE statement encountered while not in a loop. (WHILE-DO)
40	Condition expected. A condition is expected but not found. (eg. before a THEN or DO reserved word a condition is expected but not found)
41	No corresponding IF statement. A THEN or ELSE reserved word is encountered without a preceding IF statement.
42	Invalid use of braces. A statement block is encountered where it is not allowed.
43	Duplicate GOTO label. The GOTO label encountered is already defined.
44	Label expected. A GOTO statement is encountered but no label is specified or the label specified contains invalid characters.
45	Colon expected. A label definition should end in a colon.
46	Label does not exist. A GOTO statement jumps to an undefined label.
47	Cannot jump into sub-statement. Invalid GOTO construction. Cannot jump into an iteration.
48	GOTO jumps to itself. Invalid goto construction. The statement following the label referenced is the GOTO statement itself.
49	No label allowed here. A label definition is encountered but not expected.
50	Ole error. An error was triggered by an OLE server.
51	User error, Sub-errorcode meaning: -1 `Fatal` called. -2 Function implementation not finished / not supported -3 "Bad parameter" passed as argument to the function -4 "Not now", function cannot be called at the moment -5 "File not found", the function cannot find the specified file -6 "Create File failed", the function cannot create the requested file -7 "Cannot write", the function cannot write to the specified file -8 "Bad index", an invalid index is specified, the specified index is out of range -9 "Bad atom", an atom object argument points to a non-allocated object (dangling pointer) -10 "Database error", a no further specified database error occured
52	The referenced object variable is nill.
53	Object: unknown method. An unknown object method is called.
54	Object: unknown property or variable.
55	Object: unknown constructor. The constructor called is unknown.
56	Object: bad parameter. The parameters are incompatble with the object method.
57	Object: bad index. An invalid index is used for an indexed object property.
58	Object: readonly property. The object property is readonly and cannot be changed.
59	Object: writeonly property. The object property is writeonly.
60	Object: property cannot be assigned a value of this type.
61	Object: property cannot be assigned to this value.
62	Object: operation not allowed at this point.
63	Object: invalid base type for this operation.

10.2. Examples

Example 1 :
A J*o*PPS tool macro to generate a bill of material for the current project. The script first checks for an open project. Then the current calculation mode is memorized and changed into `CALCMODE_PROJECT`.

All reports are untagged except for the bill of material.

Results are then updated, the memorized calculation mode is restored, and the bill of material
is loaded into the viewer.

Finally the user is asked to print the result.

Add the script to the J*o*PPS Tools menu. (see *3.2 Tool macro's*)

```
IF GetActiveProjectIndex() < 0 THEN
  Fatal("No project loaded !");

OldCalcMode := GetCalcMode();
Recalc := !ResultsValid();

IF OldCalcMode <> CALCMODE_PROJECT THEN
{
  SetCalcMode(CALCMODE_PROJECT);
  Recalc := TRUE;
};

TagReport(SLOT_ALL,FALSE);
TagReport(SLOT_BILL_OF_MATERIAL,TRUE);
IF Recalc THEN Calculate() ELSE UpdateReports();
SetCalcMode(OldCalcMode);

IF HasResult(SLOT_BILL_OF_MATERIAL) THEN
{
  IF ReportInViewer() <> SLOT_BILL_OF_MATERIAL THEN
    ShowResult(SLOT_BILL_OF_MATERIAL);

  IF AskYN('Print result ?') THEN
    PrintResult(False,0);
}
ELSE
{
  Fatal("No result !");
};
```

Example 2 :
A JoPPS tool macro to generate an offer in Microsoft Word for the current project.

The example is similar to example 1 but now the user is asked to run the generated offer script. Doing so will produce an offer in MS Word.

The J*o*PPS "OFFER" module and Microsoft Word 97 Professional (or better) are required.
Add the script to the J*o*PPS Tools menu. (see `3.2` *Tool macro's*)

```
IF GetActiveProjectIndex() < 0 THEN
  Fatal("No project loaded !");

OldCalcMode := GetCalcMode();
Recalc := !ResultsValid();

IF OldCalcMode <> CALCMODE_PROJECT THEN
{
  SetCalcMode(CALCMODE_PROJECT);
  Recalc := TRUE;
};

TagReport(SLOT_ALL,FALSE);
TagReport(SLOT_OFFER_SCRIPT,TRUE);
IF Recalc THEN Calculate() ELSE UpdateReports();
SetCalcMode(OldCalcMode);

IF HasResult(SLOT_OFFER_SCRIPT) THEN
{
  IF ReportInViewer() <> SLOT_OFFER_SCRIPT THEN
    ShowResult(SLOT_OFFER_SCRIPT);

  IF AskYN('Start WORD ?') THEN
  {
 JOPPS.RunReport(SLOT_OFFER_SCRIPT,RUN_RUN);
  };
}
ELSE
{
  Fatal("No result !");
};
```

Example 3 :
A J*o*PPS tool macro to generate a batch optimization list for the projects currently open.

The following example will output an optimization list for all open projects.
The batch reference used it the name of the first project in the projectpool. The batch parameter dialog is not displayed if only one project is open.

Add the script to the J*o*PPS Tools menu. (see `3.2` *Tool macro's*)

```
IF GetActiveProjectIndex() < 0 THEN
  Fatal("No projects loaded !");

/* get batch reference */
batch := Upper(ChangeFileExt(ExtractFilename(GetProjectFilename(0)),''));

/* perform batch calculations */
SetCalcMode(CALCMODE_BATCH);
SetUI(ProjectCount()>1); /* show dialog when multiple projects */
SetBatchParams(batch,1,1,16);
TagReport(SLOT_ALL,FALSE);
TagReport(SLOT_OPTIMIZATION,TRUE);
Calculate();

/* show result */
IF HasResult(SLOT_OPTIMIZATION) THEN
{
  IF ReportInViewer() <> SLOT_OPTIMIZATION THEN
    ShowResult(SLOT_OPTIMIZATION);
}
ELSE
{
  Fatal("No result !");
};
```

Example 4 : A JoPPS report macro that loads the HTML offer result into Microsoft Word.

The example is a report script intended to load the result of the default HTML offer report into MS Word. It requires "SaveToDisk" to be enabled.

The J*o*PPS "OFFER" module and MS Word 97 Professional (or better) are required.

To turn this script into a report script refer to *3.3 Report macro's.*
This script can also be used as tool script.

```
IF !GetSaveToDisk() THEN
{
  Beep();
  IF !AskYN("SaveToDisk is not active, continue ?") THEN Halt;
};
fn := InterpreteString("%REPORTDOC%");
IF !FileExists(fn) THEN Fatal("File <"+fn+"> not found !");

word := start('word.application');
word.Visible := True;
word.Documents.Open(fn);
```

Example 5 :
Copy all records from the client table to Microsoft Outlook

The example presented here will copy all records in the contacts (clients) table to a folder in Microsoft Outlook.

Add the script to the JoPPS Tools menu. (see 3.2 Tool macro's)

```
pf := 'Personal Folders';

outlook := Start("Outlook.Application");
IF IsIDispatch(outlook) THEN
{
  mapi := outlook.GetNameSpace('MAPI');
  i := 1; toremove := 0;

  /* remove JoPPS folder if exists.. */
  WHILE i < mapi.Folders(pf).Folders.Count+1 DO
  {
    fld := mapi.Folders(pf).Folders(i);
    IF fld.Name = 'JoPPS' THEN
    {
      toremove := i; Break;
    };
    i := i + 1;
  };

  IF toremove > 0 THEN mapi.Folders(pf).Folders.Remove(toremove);

  /* recreate JoPPS folder */
  olJopps := mapi.Folders(pf).Folders.Add('JoPPS',10);
  IF !IsIDispatch(olJopps) THEN Halt;

  /* transfer clients */
  client := Contacts.Create();
  client.First();
  WHILE !client.Eof DO
  {
    contact := olJopps.Items.Add();
    contact.Fullname := client.Code;
    contact.FirstName := client.Code;
    contact.Lastname := client.Contact;
    contact.CompanyName := client.Desc;
    contact.BusinessAddressStreet := client.Address;
    contact.BusinessAddressPostOfficeBox := client.PoBox_Address;
    contact.BusinessAddressCity := client.Place;
    contact.BusinessAddressState := client.Country;
    contact.BusinessAddressPostalCode := client.Zip;
    contact.BusinessTelephoneNumber := client.Phone;
    contact.Business2TelephoneNumber := client.Phone2;
    contact.Email1Address := client.email;
    contact.Email2Address := client.email2;
    contact.Email3Address := client.email3;
    contact.Account := client.Account;
    contact.Save();
    client.Next();
  };
  client.Free();
};
```

Example 6 : Adding a new group to the current project.

This example will add a fixed frame to the current project. We assume we have a project open before starting this script.

Add the script to the JoPPS Tools menu. (see 3.2 Tool macro's)

```
IF AddAssembly('NEW') THEN
{
  curgroep := GetCurrentAssembly();
  frame := curgroep.Children[0];

  elmt := FrameElement.Create(frame,ELMTKIND_OUTERFRAME,1);
  elmt.Profile.System            := 'TS50';
  elmt.Profile.Code              := '136';
  elmt.Finish                    := '51';
  elmt.Definition.From.Code.Kind := ELMTKIND_FICTIF;
  elmt.Definition.From.Code.Id   := 1;
  elmt.Definition.From.Numerator := 99;
  elmt.Definition.From.Divisor   := 99;
  elmt.Definition.From.Measure   := 0.0;
  elmt.Definition.Till.Code.Kind := ELMTKIND_FICTIF;
  elmt.Definition.Till.Code.Id   := 1;
  elmt.Definition.Till.Numerator := 0;
  elmt.Definition.Till.Divisor   := 99;
  elmt.Definition.Till.Measure   := 0.0;
  elmt.Rebuild(False);

  elmt := FrameElement.Create(frame,ELMTKIND_OUTERFRAME,3);
  elmt.Profile.System            := 'TS50';
  elmt.Profile.Code              := '136';
  elmt.Finish                    := '51';
  elmt.Definition.From.Code.Kind := ELMTKIND_FICTIF;
  elmt.Definition.From.Code.Id   := 3;
  elmt.Definition.From.Numerator := 0;
  elmt.Definition.From.Divisor   := 99;
  elmt.Definition.From.Measure   := 0.0;
  elmt.Definition.Till.Code.Kind := ELMTKIND_FICTIF;
  elmt.Definition.Till.Code.Id   := 3;
  elmt.Definition.Till.Numerator := 99;
  elmt.Definition.Till.Divisor   := 99;
  elmt.Definition.Till.Measure   := 0.0;
  elmt.Rebuild(False);

  elmt := FrameElement.Create(frame,ELMTKIND_OUTERFRAME,2);
  elmt.Profile.System            := 'TS50';
  elmt.Profile.Code              := '136';
  elmt.Finish                    := '51';
  elmt.Definition.From.Code.Kind := ELMTKIND_OUTERFRAME;
  elmt.Definition.From.Code.Id   := 1;
  elmt.Definition.From.Numerator := 0;
  elmt.Definition.From.Divisor   := 99;
  elmt.Definition.From.Measure   := 0.0;
  elmt.Definition.Till.Code.Kind := ELMTKIND_OUTERFRAME;
  elmt.Definition.Till.Code.Id   := 3;
  elmt.Definition.Till.Numerator := 0;
  elmt.Definition.Till.Divisor   := 99;
  elmt.Definition.Till.Measure   := 0.0;
  elmt.Rebuild(False);

  elmt := FrameElement.Create(frame,ELMTKIND_OUTERFRAME,4);
  elmt.Profile.System            := 'TS50';
  elmt.Profile.Code              := '136';
  elmt.Finish                    := '51';
  elmt.Definition.From.Code.Kind := ELMTKIND_OUTERFRAME;
  elmt.Definition.From.Code.Id   := 1;
  elmt.Definition.From.Numerator := 99;
  elmt.Definition.From.Divisor   := 99;
  elmt.Definition.From.Measure   := 0.0;
  elmt.Definition.Till.Code.Kind := ELMTKIND_OUTERFRAME;
  elmt.Definition.Till.Code.Id   := 3;
  elmt.Definition.Till.Numerator := 99;
  elmt.Definition.Till.Divisor   := 99;
  elmt.Definition.Till.Measure   := 0.0;
  elmt.Rebuild(False);
  open :=
         FrameOpening.Create(frame,frame.Definition.X + frame.Definition.Width /
         2,frame.Definition.Y + frame.Definition.Height / 2);
  open.Rebuild(False);
}
ELSE
{
 ShowMessage('Assembly 'NEW' already exists!');
}
```

Example 7 :
Autosave settings.

Example for manipulation of autosave settings.

```
/* Get autosave settings */
bAutoSaveOnTimer  := GetAutoSaveTimer();
aInterval         := GetAutoSaveInterval();
bAutoSaveOnClose  := GetAutoSaveOnClose();
bBackupBeforeSave := GetAutoBackup();
bAutoSaveOnCreate := GetAutoSaveOnCreate();
bAutoRecover      := GetAutoRecover();

/* Set autosave settings */
bAutoSaveOnTimer := True;
SetAutoSaveTimer(bAutoSaveOnTimer);
aInterval := 10;
SetAutoSaveInterval(aInterval);
SetAutoSaveOnClose(bAutoSaveOnClose);
SetAutoBackup(bBackupBeforeSave);
SetAutoSaveOnCreate(bAutoSaveOnCreate);
SetAutoRecover(bAutoRecover);

/* Persist autosave settings */
PersistAutoSave();
```

10.3. Adding your own functions to JoPPS-Script

You can add your own functions to J*o*PPS-Script by writing a Dynamic Link Library (DLL) .

The DLL should be called JSEXT.DLL and reside in the J*o*PPS program directory where it is loaded automatically when J*o*PPS starts.

The library contains a number of exported routines called by the J*o*PPS-Script interpreter.

The loading of the JSEXT.DLL can be disabled by specifying the **-NOJSEXT** parameter at the command line.

We will not discuss the inner workings of the interpreter and its interaction with JSEXT.DLL in detail, instead we present an example of a working JSEXT.DLL example written in Borland Delphi 4.

The following Delphi 4 example "learns" J*o*PPS-Script the function SUM :

```
library jsext;

{$A+,B-,C-,D-,E-,F-,G+,H+,I-,J+,K-,L-,M-,N+,O-,P+,Q+,R+,S-,T-,U+,V+,W-,X+,Y-,Z1}

////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//  The JoPPS-Script function SUM calculates the sum of all D-type            //
//  parameters passed. S and I-type parameters are ignored.                   //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////

uses
  ShareMem;

{$R *.RES}

function DLL_INITIALIZE (sp : Pointer; current_db_version : Integer) : Integer;
begin Result := 0; end;

function DLL_INFORMATION(sp : Pointer; dllInfo : Pointer) : Integer;
begin Result := 0; end;

function DLL_TERMINATE (sp : Pointer) : Integer;
begin Result := 0; end;

procedure JS_OnExecComplexFunction (
        numvars    : Integer;
  const varparams  : array of variant;
  const vartypes   : String;
        numparams  : Integer;
  const funcparams : array of Variant;
  const paramtypes : String; var Result : Variant);

  var i : Integer; funcName : String; sum : Double;

begin
  if (numvars = 1) and (vartypes = 'V') then begin
    funcName := varparams[0];
    if funcName = 'SUM' then begin
      sum := 0.0;
      for i := 0 to numparams-1 do
        if paramtypes[i+1]='D' then sum := sum + funcparams[i];
      Result := sum;
    end;
  end;
end;

exports
  DLL_INITIALIZE           ,
  DLL_INFORMATION          ,
  DLL_TERMINATE            ,
  JS_OnExecComplexFunction ;
end.
```