PRO Suite

Plugin Developer's Guide

VDOMBOX INTERNATIONAL

Contents

Introduction

This document contains the information about how to enhance the functionality of the ProSuite applications with the help of Plugins. This description is specially made for users who have different level of programming skills: from beginners to advanced. Such topics as the process of plugins creating, way of using the VScript programming language for writing plugins and also the description of all objects of ProSuite applications are covered in the document.

ProSuite applications represent web applications which are aimed at making the collaborative work easier and improving the quality of job execution. It consists of separate web applications which can both work independently and interact with each other.

The opportunity of creating plugins for ProSuite is offered in order to allow users to adjust the ProSuite web applications to executing the specific tasks of some company. Plugin is an additional programming module built in the ProSuite application, that enhances the functional capabilities of this application.

In the next sections of this document you will find the description of how plugins work and what they consist of, what scripting objects and the objects of applications are and how they can be used in the macro's source code, and how to create plugins for the ProSuite applications.

1 ProSuite plugins overview

Plugin represents an additional module, that provides to a ProSuite application an additional or customized functionality. All plugins for ProSuite applications are written in the VScript programming language, that has much in common with the VBScript language. On the whole, plugins represent XML files, what implies the possibility of exporting and importing plugins. This feature gives the opportunity of copying plugins from one server to another one with the same application installed, so the developer can create plugins not only for the needs of his company but also to the orders of customers.

Since plugins for ProSuite represent XML files, they can be created both in the ProSuite application and exported to it. The constituent part of a plugin is macro, which executes the main actions. Plugins can also include timers, which trigger the macros execution on the expiry of the specified period of time.

Macro is a part of plugin that contains the source code executing some actions. Depending on the plugin type there can be distinguished three types of macros: button, event and timer.

When you create a **button macro**, a button is added to the Plugins section of the application, that executes the macro's source code. The plugins, in which such type of macro is used, allow user to manipulate the objects of the application's page.

The code of the **event macro** is executed when the specified event occurs, for example, this event can be like adding a file to the Files of ProShare, editing the contact in ProContact and deleting the calendar in ProPlanning. The list of possible event of each ProSuite application you can find in the section where all objects of application are described.

The macro, which code is executed on the expiry of the specified period of time, is called **timer macro**. The period of time is specified by the timer, which is bound to the macro.

There is one more type of macro - a **configuration macro**. When you add a macro of such type, a Configure plugin button is added to the plugin. When you press this button the source code of the macro is executed. Though a configuration macro is similar in nature to the button macro, it provides an opportunity of configuring the plugin to which this macro refers.

Timer is a part of plugin that is used to count the specified time intervals, for executing a macro's source code. On the expiry of the specified period of time, the special event occurs which triggers the execution of the macro.

2 VScript as plugins developing language

VScript is a programming language of macros for the plugins of the ProSuite applications. When writing code in VScript you can use the objects of different levels, such as:
global objects - the global objects which are used to call the server's data;
common objects - the objects common for all ProSuite applications;
application objects - the objects which are peculiar only for one ProSuite application, like the objects of ProPlanning, the objects of ProContact and so on.

The description of the VScript programming language and the objects of different levels you can find in the sections below.

2.1 VScript language reference

2.1.1 VScript data type

A variable is a named storage location holding the data that can change during the execution of the VScript program. It is used for representing the data types. Variable names must begin with an alphabetic character, must be unique within the same scope, and cannot be longer than 255 characters.

A variable in VScript represents a Variant data type. The different categories of data that variables hold can be classified into subtypes. A Variant provides a uniform programming interface for variable's subtypes (such as integer, real and so on) which get determined at runtime. Since Variant is the only data type in VScript, it is also the data type returned by all functions in VScript.

A Variant can contain different kinds of information, depending on how it is used. For example, the Variant can contain either numeric or string information, it behaves as a number when you use it in a numeric context and as a string when you use it in a string context. That is, if you are working with the data that looks like numbers, VScript assumes that these data are numbers and does what is most appropriate for numbers. Similarly, if you are working with data that can only be string data, VScript treats it as string data.

Beyond the simple numeric or string classifications, a Variant can make further distinctions about the specific nature of numeric information. For example, you can have numeric information that represents a date or a time. When used with other date or time data, the result is always expressed (with a little exceptions) as a date or a time. You can also have a rich variety of numeric information ranging in size from Boolean values to huge floating-point numbers. Most of the time, you can just put the kind of data you want in a Variant, and the Variant behaves in a way that is most appropriate for the data it contains.

There are the following subtypes in VScript:

Subtype	Description
Array	A complex subtype containing an array data.
	Dim MyArray(9) Array(1, 2, 3)
Binary	A simple subtype containing the binary data.
Boolean	A simple subtype representing a logical data type which can contain either True (equivalent to 0) or False (equivalent to -1).
	True

	False
Date	A complex subtype containing an integer that represents date (between January 1, 100 to December 31, 9999) and time. #01.01.2010# #18:20#
Dictionary	A simple subtype containing specific key-value pairs. Dictionary(1, "A", 2, "B", 3, "C")
Double	A simple subtype containing a real number (8 bytes) with the precision 15 digits (double-precision) in floating-point format ranging from -1.79769313486232e308 to -4.94065645841247e-324 for negative values, and from 4.94065645841247e-324 to 1.79769313486232e308 for positive values. 1.25 1E-2
Empty	A simple subtype that is used as the initial variable's value or as default value of variables which are not initialized explicitly. Empty
Error	A complex subtype used for storing exceptions. DivisionByZero SubscriptOutOfRange
Generic	A complex subtype that is a parent object for all complex objects, both user's and system's ones.
Integer	A simple subtype containing an integer number (4 bytes) in the range from -2,147,483,648 to 2,147,483,647. A value can be only a decimal figure. 123 -567
Nothing	A complex subtype showing the absence of object. It is the analogue of Empty subtype, but is used for complex objects, for example, when it is necessary to clear the complex object. Nothing
Null	A complex subtype intentionally containing no valid data. It is usually used in databases when it is necessary to show the absence of data. Null
String	A simple subtype containing a variable-length string that can be up to approximately 2 billion characters in length in the Unicode character set. "This is a string"

2.1.2 VScript operators

Operator represents a command which performs an operation on one or more code elements - **operands**. VScript has a full range of operators, including arithmetic, comparison, logical (or relational), assignment, membership and identity ones. Operators combined with the code returning some value (for example, constants or variables) form **expressions** or **statements** (in case of the assignment operator).

There is no limit of the number of operators which can be combined into expression, but understanding of operator precedence is necessary to ensure that you will get the results which you are expecting.

If in an expression several operations occur at the same time, each part of the expression is evaluated and resolved in a predetermined order called **operator precedence**.

There follows some tips of how expressions will be evaluated and how you can change the order of evaluating them:
To override the order of precedence and force some parts of an expression to be evaluated before others you can use parentheses. Operations within parentheses are always performed before those outside. Within parentheses, however, standard operator precedence is maintained;
If there are operators from several categories in one expression, the operators are evaluated in the following order (beginning from the highest precedence to the lowest):
arithmetic operators and concatenation operators are evaluated in the order of precedence described in the next sections;
comparison operators - have equal precedence; that is, they are evaluated in the left-to-right order in which they appear;
logical operators - are evaluated in the order of precedence described in the next sections;
Assignment operators compose a stand-alone group of operators, they do not follow the general rules of precedence;
If there are two or more operators in a statement of the equal precedence, they are evaluated in the left-to-right order in which they appear.

2.1.2.2 Assignment operators

The assignment operator allows you to assign new values to variables or object properties. The assignment operator takes the value on the right side of the operator and assigns it to the variable on the left side. There is only one assignment operator in VScript.

Operator	Name	Description	Example
=	simple assignment	Assign values from right side operands to left side operand.	c=a+b

2.1.2.2 Arithmetic operators

The arithmetic operators perform the arithmetic operations that involve the calculation of numeric values represented by constants, variables, other expressions, function and property calls. These operators have usual mathematical precedence.

Operator	Name	Description	Example
^	exponentiation	Raise a number to the power of an exponent.	a^b
-	unary negation	Indicate the negative value of a numeric operand.	-a
*	multiplication	Multiply two numeric operands.	a*b
/	division	Divide two numeric operands and return a floating-point result.	b/a
\	integer division	Divide two numeric operands and return an integer result.	a\b
Mod	modulus arithmetic	Divide two numeric operands and return the remainder.	b Mod a
+	addition	Sum two numeric operands.	a+b
-	subtraction	Find the difference between two numeric operands.	a-b

2.1.2.3 Concatenation operators

C Concatenation operator joins multiple strings into a single string. There is only one concatenation operator in VScript. It is not recommended to use + operator to concatenate strings, because it can bring out some unpredictable results.

Operator	Name	Description	Example
&	string concatenation	Concatenate string operands. It is defined exclusively for strings and reduces the chances of generating an unintended conversion.	a&b

2.1.2.4 Comparison operators

Comparison operators are used to compare two expressions, or in case of **Is** operator, to compare two object reference variables. The return value of such operation is a Boolean value that represents the result of the comparison. All kinds of objects can be compared, no matter what information they contain. All comparison operators have the same precedence (which is higher than that of the Boolean operations).

Operator	Name	Description	Example
=	equal to	Compare two operands and return a Boolean value (*True* or *False*) as to the validity of the comparison.	a=b
<>	not equal to	Compare two operands and return a Boolean *True* if the operands are not equal.	a<>b
<	less than	Compare two operands and return a value of *True* if the left operand is less than the value of the right operand.	a<b
>	greater than	Compare two operands and return a value of *True* if the left operand is greater than the value of the right operand.	a>b
<=	less than or equal to	Compare two operands and return *True* if the first operand is less than or equal to the second.	a<=b
>=	greater than or equal to	Compare two operands and return *True* if the first operand is greater than or equal to the second.	a>=b
Is	object equivalence	Compare two operands and evaluate to *True* if the variables on either side of the operator point to the same object and *False* otherwise. This operator can work incorrectly with VScript additional objects (see the section 2.2 VScript additional objects).	a Is b Is results in 1 if id(x) equals id(y).
IsNot	object inequivalence	Compare two operands and evaluate to *False* if the variables on either side of the operator point to the same object and *True* otherwise. This operator can work incorrectly with VScript additional objects (see the section 1.2 VScript additional objects).	x IsNot y IsNot results in 1 if id(x) is not equal to id(y).

2.1.2.5 Logical operators

The logical operators are used for logical combinations in comparisons. They compare Boolean expressions and return a Boolean result. Any object can be tested for truth value, for use in an if- or while-condition or as operand of the Boolean operations. Using parentheses you can combine several different logical operations in one more complex operation. The **And**, **Or** and **Xor** operators take two operands, and the **Not** operator takes a single operand.

Operator	Name	Description	Example
Not	logical negation	Reverse the logical state of the operand. If a condition is *True*, Not operator will make *False*.	Not a
And	logical conjunction	If both operands are *True*, the condition becomes *True*.	a And b
Or	logical disjunction	If any of the two operands is nonzero then then condition becomes *True*.	a Or b
Xor	logical exclusion	Perform a logical exclusion on two Boolean operands, or a bitwise exclusion on two numeric operands.	a Xor b

2.1.3 VScript variables

A variable is a named storage location holding the data that can change during the execution of the VScript program. It is used for representing any data type. Variable names are case insensitive, must begin with an alphabetic character, must be unique within the same scope, cannot be longer than 255 characters.

2.1.3.1 Explicit declaring variables (Dim statement)

This way is a recommended one. Explicit declaring a variable means that you allocate the storage space for a variable that will be used somewhere in your code. **Dim** statement declares and allocates storage space for one or more variables. With the help of this statement the variables are declared explicitly in your script.

Explicit declaring of a variable

```
Dim DegreesFarenheit
```

You can declare multiple variables by separating each variable name with a comma.

Explicit declaring of multiple variables

```
Dim Top, Bottom, Left, Right
```

2.1.3.2 Implicit declaring variables (assignment operator)

A variable can be declared implicitly inside the code. It is implemented by simple assigning a value to a variable with the help of assignment operators. The expression created should be as follows: the variable is on the left side of the expression and the value you want to assign to the variable is on the right as in the many other languages.

In the following example, the simple assignment operator (=) is used:

Implicit declaring of a variable

```
B = 200
```

2.1.3.4 Scalar and array variables

All variables can be divided into two groups: scalar and array ones. They are declared in the same way, except that the declaration of an array variable uses parentheses ( ) following the variable name. A **scalar variable** is a variable containing a single value. An **array variable** can contain many indexed values; their counting starts from 0 (zero-based arrays). The array is declared within a procedure using either **Dim** or **ReDim** statement. Array variables can be static and dynamic.

Static array has a fixed size, because we can definitely say how many elements it includes.

Declaring a static array

```
Dim MyArray(5)
```

MyArray in this example contains 6 elements (counting the elements starts from 0). You can assign data to each of the elements of the array using an index into the array. Beginning from zero and finishing with 5, the data can be assigned to the elements of an array.

Assigning values to the elements of an array

```
MyArray(0) = "a"
MyArray(1) = "b"
MyArray(2) = "c"
...
MyArray(5) = "f"
```

The data can be retrieved from any element using an index into the particular array element you want.

Assigning a value of an array's element to a variable

```
...
```

```
SomeVariable = MyArray(5)
...
```

An array can have from 1 to 60 **dimensions**. A dimension is a direction in which you can vary the specification of an array's elements. The number of dimensions an array has is called its **rank**. To declare multiple dimensions you must separate an array's size numbers in the parentheses by commas.

Declaring an array with multiple dimensions

```
Dim MyArray (3, 7)
```

Dynamic array is an array whose size changes during the time when the script is running. For an array to be dynamic, no size or number of dimensions is placed inside the parentheses. There is no limit to the number of times you can resize a dynamic array, although if you make an array smaller, you will lose the data in the eliminated elements.

Declaring a dynamic array

```
Dim MyArray()
```

In the example below the **ReDim** statement sets the initial size of the dynamic array to 5.

Resizing the dynamic array

```
ReDim MyArray(5)
```

A subsequent **ReDim** statement resizes the array to 10.The **Preserve** statement is used to preserve the contents of the array as the resizing takes place.

Resizing the dynamic array with saving the contents

```
ReDim Preserve MyArray(10)
```

2.1.4 VScript constants

A constant is a named storage location holding the data that cannot change during the execution of the VScript program. It raises an error on trying to rewrite it. It is a meaningful name that takes the place of a number or string and raises an error on trying to rewrite it.

A constant can include a string literal, a numeric literal, date and time literals, or any combination that includes arithmetic or logical operators except **Is** and exponentiation. Additional constants can be defined by the user with the **Const** statement. You can use constants anywhere in your code in place of actual values.

You can create user-defined constants in VScript using the **Const** statement. With the help of this statement, you can create string or numeric constants with meaningful names and assign them literal values.

Different types of values are assigned in different ways:
numeric values - simple assignment;
string values - a value is enclosed in quotation marks (" ");
date literals and time literals - a value enclosed in number signs (#).

Declaring different types of constants

```
Const MyStringConst = "A string constant."
Const MyIntConst = 1999
Const MyDateConst = #6-1-97#
Const MyTimeConst = #1:10:00 AM#
```

You may want to adopt a naming scheme to differentiate constants from variables. This will prevent you from trying to reassign constant values while your script is running. For example, you might want to use a prefix on your constant names, or you might name your constants in all capital letters. Differentiating constants from variables eliminates confusion as you develop more complex scripts.

2.1.5 VScript procedures

A **procedure** is a part of code grouped into a single entity. Procedures have such feature as reusability. Once you create a procedure, you can use its code by calling that procedure. Two kinds of VScript procedures - the **Sub** procedure and the **Function** procedure, are described below.

Every piece of data is passed to the procedures using arguments (constants, variables, or expressions that are passed by a calling procedure). They serve as placeholders for the data you want to pass into the procedure. When you create a procedure using either the **Sub** statement or the **Function** statement, parentheses must be included after the name of the procedure. The arguments are placed inside the parentheses, separated by commas. If a procedure has no arguments, its **Sub** or **Function** statement must include an empty set of parentheses ().

2.1.5.1 Sub procedure

A **Sub** procedure can perform some actions, but cannot return values. A **Sub** procedure must be enclosed in **Sub** and **End Sub** statements.

To create a **Sub** procedure use the following syntax:

```
Sub name [(arglist)]
    [statements]
    [Exit Sub]
    [statements]
End Sub
```

In the following example a **Sub** procedure NewTotal is declared.

Declaring a Sub procedure

```
Sub NewTotal
   Dim myNum, total
   For myNum = 16 To 2 Step -2
      total = total + myNum
   Next
   this.Label.value = "The total is " & total
End Sub
```

The example of declaring the **Sub** procedure that converts temperature.

Declaring a Sub procedure

```
Sub ConvertTemp
   temp = request.QueryString("Degrees")
   this.Label.value =_
      "The temperature is " & Celsius(temp) &  " degrees C. "
```

```
End Sub
```

2.1.5.2 Function procedure

A **Function** procedure is similar to a **Sub** procedure but it cannot only perform some actions but also return values. A **Function** procedure returns a value by assigning a value to its name in one or more statements of the procedure. The returned data type of a **Function** procedure is always a Variant. A **Function** procedure must be is enclosed in **Function** and **End Function** statements.

To create a **Function** procedure use the following syntax.

Scheme of declaring a Function procedure

```
Function name [(arglist)]
    [statements]
    [name = expression]
    [Exit Function]
    [statements]
    [name = expression]
End Function
```

In the following example, fDegrees is a placeholder for the value passed into the Celsius function for conversion.

Declaring a Function procedure

```
Function Celsius(fDegrees)
   Celsius = (fDegrees -32) * 5 / 9
End Function
```

2.1.5.3 Calling procedures from code

A **Function** in a code must always be used on the right side of a variable assignment or in an expression.

Calling a Function procedure

```
Temp = Celsius(fDegrees)
Message = "The temperature is " & Celsius(temp) & " degrees C. "
```

To call a **Sub** procedure from another procedure, type the name of the procedure along with values for any required arguments, each separated by a comma. The **Call** statement is not required, but if you use it, you must enclose the arguments in parentheses.

The following example shows two calls to the MyProc procedure. One uses the **Call** statement in the code; the other does not. Both do exactly the same thing.

Calling a Sub procedure

```
Call MyProc(firstarg, secondarg)
MyProc firstarg, secondarg
```

Note, that the parentheses are omitted when the Call statement is not used.

2.1.7 VScript flow control

The code of a script that you have written is executed in a certain order when you run it. The order of code execution is called **flow**. Usually the script engine starts with the first line of the script, executes it, then goes to the next line and so on, until it reaches the end.

VScript also provides the opportunity of flow control: it can be done with the help of conditional statements (branching) and looping statements. These control flow statements can alter the way of flow execution by branching to other code sections (in case of the conditional statements) or repeating some code sections (in case of the looping statements).

2.1.7.1 Conditional statements

The conditional statements provide an opportunity of code branching. There are two cases of using the conditional statements in VScript.

2.1.7.1.1 If...Then...Else

If...Then...Else statement conditionally executes a group of statements, depending on the value of an expression (*True* or *False*). Usually the condition is an expression that uses a comparison operator to compare one value or variable with another. **If...Then...Else** statement can include as many levels as you need.

End If terminates the **If...Then...Else** block. **ElseIf** clauses expand the functionality of the statement so you can control program flow based on different possibilities.

The single-line syntax is used for short, simple tests.

Scheme of single-line using the statement

```
If condition Then [ statements ] [ Else [ elsestatements ] ]
```

The multiple-line syntax provides more structure and flexibility and is usually easier to read, maintain, and debug:

Scheme of multiple-line using the statement

```
If condition Then
   [ statements ]
[ ElseIf elseifcondition Then
   [ elseifstatements ] ]
[ Else
   [ elsestatements ] ]
End If
```

For example, to run one statement if a condition is *True*, use the single-line syntax. Note that this example omits the **Else** keyword.

Running the single-line statement if a condition is True

```
Sub FixDate
   Dim myDate
   myDate = #2/13/95#
   If myDate < Now Then myDate = Now
End Sub
```

To run more than one line of code if a condition is *True*, use the multiple-line syntax. This syntax includes the **End** If statement, as it is shown in the following example.

Running the multiple-line statement if a condition is True

```
Sub AlertUser(value)
```

```
   If value = 0 Then
      this.Label.color = "FF0000"
      this.Label.Font = "bold 12pt"
   End if
End Sub
```

You can use an **If...Then...Else** statement to define two blocks of executable statements: one block to run if the condition is *True*, the other block to run if the condition is *False*.

Running one statement if a condition is True and another if a condition is False

```
Sub AlertUser(value)
   If value = 0 Then
       this.Label.color = "FF0000"
       this.Label.Font = "bold 12pt"
   Else
      this.Label.color = "FFFF00"
      this.Label.Font = "bold 10pt"
   End if
End Sub
```

2.1.7.1.2 Select...Case

Select...Case statement allows selecting from several alternatives. It runs one of several groups of statements, depending on the value of an expression. The **Select Case** construction starts and the **End Select** terminates the execution of the **Select...Case** statement.

To create a **Select...Case** statement use the following syntax:

Scheme of using the statement

```
Select Case testexpression
    [Case expressionlist-n
       [statements-n]] . . .
    [Case Else expressionlist-n
       [elsestatements-n]]
End Select
```

A **Select...Case** statement works with a single test expression that is evaluated once, at the top of the structure. The result of the expression is then compared with the values for each **Case** in the structure. If there is a match, the block of statements associated with that **Case** is executed, as in the following example.

Running the statement

```
Select Case Request.Form("CardType")
   Case "MasterCard"
      DisplayMCLogo
      ValidateMCAccount
   Case "Visa"
      DisplayVisaLogo
      ValidateVisaAccount
  Case "American Express"
      DisplayAMEXCOLogo
      ValidateAMEXCOAccount
  Case Else
      DisplayUnknownLogo
      PromptAgain
```

```
End Select
```

In comparison with the **If...Then...Else** statement which can evaluate different expressions for each **ElseIf** statement, the **Select...Case** statement evaluates an expression written at the top of the structure.

2.1.7.2 Looping statements

Looping allows you to run a group of statements repeatedly. Some loops repeat statements until a condition is *False*; others repeat statements until a condition is *True*. There are also loops that repeat statements a specific number of times. There are several looping statements available in VScript.

2.1.7.2.1 Do...Loop

You can use **Do...Loop** statement to run a block of statements an indefinite number of times. The **Do...Loop** statement repeats a block of statements either while a Boolean condition is *True* or until a condition becomes *True*.

To create a **Do...Loop** statement use the syntax:

Scheme of using the statement with checking the condition before the statement

```
Do [{While | Until} condition]
    [statements]
    [Exit Do]
    [statements]
Loop
```

Or, you can use the following syntax:

Scheme of using the statement with checking the condition after the statement

```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

Use the **Do...Loop** statement to repeat it while a condition is *True*. To check a condition in a **Do...Loop** statement use the **While** keyword. You can check the condition before you enter the loop as it is shown in the following example. In this example if myNum is set to 9 instead of 20, the statements inside the loop will never run.

Checking the condition before the statement

```
Sub ChkFirstWhile()
   Dim counter, myNum
   Counter = 0
   myNum = 20
   Do While myNum > 10
      myNum = myNum - 1
      counter = counter + 1
   Loop
   this.Label.value =_
      "The loop made " & counter & " repetitions."
End Sub
```

You can check the condition after the loop has been run at least once as it is shown in the example. In this example the statements inside the loop run only once because the condition is already *False*.

Checking the condition after the statement

```
Sub ChkLastWhile()
   Dim counter, myNum
   Counter = 0
   myNum = 9
   Do
      myNum = myNum - 1
      counter = counter + 1
   Loop While myNum > 10
   this.Label.value =_
      "The loop made " & counter & " repetitions."
End Sub
```

Do…Loop statement can also be used to repeat it until a condition becomes *True*. There are two ways to use the **Until** keyword to check a condition in a **Do...Loop** statement. You can check the condition before you enter the loop as it is shown in the following example.

Looping until a condition becomes True

```
Sub ChkFirstUntil()
   Dim counter, myNum
   Counter = 0
   myNum = 20
   Do Until myNum = 10
      myNum = myNum + 1
      counter = counter + 1
   Loop
   this.Label.value =_
      "The loop made " & counter & " repetitions."
End Sub
```

You can check the condition after the loop has run at least once as it is shown in the example. As long as the condition is *False*, the looping occurs.

Looping as long as the condition is False

```
Sub ChkLastUntil()
   Dim counter, myNum
   Counter = 0
   myNum = 1
   Do
      myNum = myNum + 1
      counter = counter + 1
   Loop While myNum = 10
   this.Label.value =_
      "The loop made " & counter & " repetitions."
End Sub
```

To exit a **Do...Loop** statement use the **Exit Do** statement. Since you usually want to exit only in certain situations, like in order to avoid an endless loop, you should use the **Exit Do** statement in the *True* statement block of the **Do...Loop** statement. If the condition is *False*, the loop runs as usual. In the following example, myNum is assigned a value that creates an endless loop. The **Do...Loop** statement checks for this condition, preventing the endless repetition.

Exiting the statement

```
Sub ExitExample()
   Dim counter, myNum
   Counter = 0
   myNum = 9
   Do Until myNum = 0
      myNum = myNum - 1
      counter = counter + 1
      If myNum < 5 Then Exit Do
   Loop
   this.Label.value =_
      "The loop made " & counter & " repetitions."
End Sub
```

2.1.7.2.2 While...Wend

A **While...Wend** statement executes a series of statements as long as a given condition is *True*. Due to the lack of flexibility in **While...Wend**, it is recommended that you use **Do...Loop** statement instead.

To create a **While...Wend** statement use the following syntax.

Scheme of using the statement

```
While condition
    [statements]
Wend
```

If condition is *True*, all statements are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and condition is again checked. If condition is still *True*, the process is repeated. If it is not *True*, execution resumes with the statement following the **Wend** statement.

2.1.7.2.3 For...Next

For...Next statement repeats a group of statements a specified number of times. In such statements you can use a counter variable whose value increases or decreases with each repetition of the loop.

To create a **For...Next** statement use the following syntax:

Scheme of using the statement

```
For counter = start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next
```

In the following example a procedure called MyProc is caused to execute 50 times. The **For** statement specifies the counter variable x and its start and end values. The **Next** statement increments the counter variable by 1.

Running one and the same statement several times

```
Sub DoMyProc50Times
   Dim x
   For x = 1 To 50
      myProc
   Next
End Sub
```

Using the **Step** keyword, you can increase or decrease the counter variable by the value you specify. In the following example, the counter variable j is incremented by 2 each time the loop repeats. When the loop is finished, the total is the sum of 2, 4, 6, 8, and 10.

Running the statement increasing the variable's value

```
Sub TwosTotal

   Dim j, total
   For j = 2 To 10 Step 2
      total = total + j
   Next
   this.Label.value = "The total is " & total
End Sub
```

To decrease the counter variable, use a negative Step value. You must specify the end value that is less than the start value. In the following example, the counter variable myNum is decreased by 2 each time the loop repeats. When the loop is finished, total is the sum of 16, 14, 12, 10, 8, 6, 4, and 2.

Running the statement decreasing the variable's value

```
Sub NewTotal()
   Dim myNum, total
   For myNum = 16 To 2 Step -2
      total = total + myNum
   Next
   this.Label.value = "The total is " & total
End Sub
```

You can exit any **For...Next** statement before the counter reaches its end value by using the **Exit For** statement. Because you usually want to exit only in certain situations, such as when an error occurs, you should use the **Exit For** statement in the *True* statement block of an **For...Next** statement. If the condition is *False*, the loop runs as usual.

2.1.7.2.4 For Each...Next

A **For Each...Next** statement repeats a group of statements for each item in a collection or each element of an array. It is similar to a **For...Next** loop, but instead of repeating the statements a specified number of times, a **For Each...Next** loop repeats a group of statements for each item in a collection of objects or for each element of an array. This is especially helpful if you do not know how many elements are in a collection.

To create a **For...Next** statement use the following syntax:

Scheme of using the statement

```
For Each element In group
    [statements]
    [Exit For]
    [statements]
Next [element]
```

Running the statement

```
For Each name In request.servervariables
   result=result & name & "-" & Resquest.servervariables(name)
Next
```

2.1.8 Handling errors

It is very difficult to prevent errors occurring when writing code. So, to develop a stable code you must manage errors inside it.

To handle the errors that may occur in a block of code, while still running code use the **Try…Catch…Finally…End Try** statement. If you suppose a particular exception might occur during a particular section of code, put the code in a **Try** block and use a **Catch** block to retain control and handle the exception if it occurs.

The syntax of the statement is the following:

Scheme of using the statement

```
Try ' Start a structured exception handler.
    [ statements ] ' Executable statements that may generate an exception in this block.
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ] ' This code runs if the statements listed in the Try block fail and the filter on the Catch statement is true.
    [ statements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ statements ] ] ' Runs before the Try statement exits.
End Try ' Finish a structured exception handler.
```

VScript provides a list of possible errors described in a table below:

Exception	Description

generic	The basic class used for all exceptions. It can be used in order to differentiate VScript errors from other errors.
	`Try`
	`    server.vscript.execute(script)`
	`except errors.generic:`
	`    print "There is a VScript error"`

property_have_no_arguments	Raised for properties if no arguments have been passed. The exception does not have additional parameters.
object_has_no_property	Raised in case of unprovided property calling. It happens, for example, when you are trying to make the property read-only, or when you are trying to assign a property, which can have a simple object as a value, using set property. It is better to designate the name of the property as its argument.
subscript_out_of_range	Raised in case of incorrect indexing during working with collections.
type_mismatch	Raised on trying to process the parameters with unexpected subtype.
wrong_number_of_arguments	Raised in functions, which accept the optional arguments, if the number of them mismatch. It accepts the function name as a parameter.
invalid_procedure_call	Raised by function in case of getting the incorrect parameters. It is better to designate the name of the function as its argument.
division_by_zero	Raised when an expression being used as a divisor has a value of zero.
overflow	Raised when you attempt an assignment that exceeds the limits of the assignment's target.

2.2 VScript built-in classes

The VScript built-in classes are used to enhance the basic functionality of code. They are accessible from any part of script and do not depend on the server's status and on other classes of environment. Besides, they can work with classes linked to some application that means, that they are accessible for the scripts executing in the context of the applications. Pay attention, that the comparison operator **Is** and **IsNot** can work incorrectly with these classes (see the section 2.1.2.4 Comparison operators).

There are built-in classes which differ in their functions:
regular expressions management;
XML support;
JSON support;
network connection support.

2.2.1 Regular expressions management

Regular expression is a class that includes other classes such as RegExp, Matches and Match. It provides simple regular expression support including declaring regular expressions, comparing, searching in a string and iterating the results.

Create an instance of the RegExp class

```
Set MyRegExp=New RegExp
MyRegExp.IgnoreCase=True
MyRegExp.Global=True
MyRegExp.Pattern="abc[a-z]+"
Set Matches=MyRegExp.Execute("klm abcdef klm defABCklm klm")
For Each Match In Matches
   Print Match.Value
Next
```

2.2.1.1 RegExp class

RegExp class has the following properties:

Property	Description
Pattern	A string defining the regular expression; it must be set before using the regular expression object.
IgnoreCase	Set or return a Boolean value indicating if a pattern search is case-sensitive or not. Default value - *False*.
Global	A Boolean value indicating that the regular expression needs being tested for matches in a string. Default value - *False*.

RegExp class has the following methods:

Method	Description
Test(string)	Return *True* if the regular expression matches the given argument (string), otherwise - *False*. `result=regex.test("w1")`
Replace(search-string, replace-string)	Replace the first argument (search-string) found in the regular expression by the second argument (replace-string). Return the first argument (search-string) if no matches are found. `result=regex.replace("w1 w2 w3","w")`
Execute(replace-string)	Similar to the Replace method, but returns a Matches collection object, containing a Match object for each successful match. It does not modify the search string. Return an empty Matches collection if no match is found. `result=regex.execute`

2.2.1.2 Matches class

Matches is a VScript built-in class representing a collection returned as a result of the Execute method. It can contain zero or more Match objects; its properties are read-only.

Matches class has the following properties:

Property	Description
Count	A value containing the number of Match objects in the collection. Read-only.
Item	A value enabling a Match objects to be randomly accessed from the Matches collection object. Read-only.

The Match objects may also be incrementally accessed from the Matches collection object, using a For Each...Next loop.

Accessing the Match objects from the Matches collection object

```
For Each Match In Matches
   Print Match.Value
Next
```

2.2.1.3 Match class

Match is a VScript built-in class that represents a subclass of the Matches class (each successful match found by the regular expression). Its properties are read-only and contain the information about each match.

Match class has the following properties:

Property	Description
FirstIndex	A value containing the position within the original string where the match occurred. This index uses a zero-based offset to record positions, meaning that the first position in a string is 0. Read-only.
Length	A value containing the total length of the matched string. Read-only.
Value	A value containing the matched value or text. It is also the default value when accessing the Match object. Read-only.

2.2.2 XML support

The XML support includes such classes used for working with XML as XMLNode, XMLAttribute, XMLElement, XMLDocument and associated exceptions.

2.2.2.1 XMLNode class

XMLNode is a VScript built-in class representing a general class that implements the XML node. XMLNode class has the following properties:

Property	Description
Parent	The parent of the current node, or Nothing for the root (document) node. Read-only.
Prev	The node that immediately precedes this one. Read-only.
Next	The node that immediately follows this one. Read-only.
Attributes	Return attributes of this node. Read-only.
Nodes	Return list of the child nodes. Read-only.
First	Return the first child node for this node. Read-only.
Last	Return the last child node for this node. Read-only.
Name	Return name of this node, if possible. Read-only.
Value	Return value of this node, if possible. Read-only.

XMLNode class has the following methods:

Method	Description
HasAttributes()	Return *True* when the node has any attributes.
HasNodes()	Return *True* when the node has any child nodes.
IsElement()	Return *True* when node is element.
IsAttribute()	Return *True* when node is attribute.
IsText()	Return *True* when this node is text.
IsCData()	Return *True* when this is CData node.
IsComment()	Return *True* when this is comment node.
IsDocument()	Return *True* when this node is root document node.
IsSameNode(Node)	Return *True* when node and this node is the same.
Append(Node)	Add new child node to this node after all other child nodes. If child node already added to another node, it is removed first.
Insert(Node, ReferenceNode)	Insert new child node to this node before reference node. If child node already added to another node, it is removed first.
Remove(Node)	Remove a child node from this node. Return node on success.
Replace(Node, ReferenceNode)	Replace an existing child node by a new one.
Normalize	Join adjacent text nodes into the single node.
Clone([Deep])	Clone the node and all child nodes if they are present. Return new node.
Compose	Compose this node in XML if possible and return string.

2.2.2.2 XMLAttribute class

XMLAttribute is a VScript built-in class representing a subclass of the XMLNode, which inherits all its properties and methods.

2.2.2.3 XMLElement class

XMLElement is a VScript built-in class representing a subclass of the XMLNode, which inherits all its properties and methods. It is an XML element with attributes, which has the following additional properties and methods.

XMLElement class has the following property:

Property	Description
Elements	Return a list of the child elements. Read-only.

XMLElement class has the following methods:

Method	Description
Search(Name)	Return a list of the child elements with specified name.
HasAttribute(Name)	Return *True* if the element has an attribute with specified name.
GetAttribute(Name)	Return element's attribute with specified name.
GetAttributeNode(Name)	Return element attribute's node.
SetAttribute(Name, Value)	Assign new value to element's attribute with specified name.
SetAttributeNode(Node)	Add a new attribute node or replace existing with the new node.
RemoveAttribute(Name)	Remove attribute from the element.

2.2.2.4 XMLDocument class

XMLDocument is an built-in class representing a subclass of the XMLElement class, which inherits all its properties and methods. It also has the following methods.

Method	Description

Method	Description
Parse(Value)	Parse string value and build XML node tree.
CreateElement(Name)	Create and return new XMLElement.
CreateTextNode(Value)	Create and return new text XMLNode with appropriate value.
CreateComment(Value)	Create and return new comment XMLNode with appropriate value.
CreateAttribute(Name)	Create and return new XMLAttribute with appropriate name.

2.2.2.5 XML exceptions

When you work with XML some exceptions can be raised. The exceptions supported in VScript are as follows:

Exception	Description
XMLError	Base exception class used for all XML exceptions.
XMLDomstirngSizeError	Raised when a specified range of text does not fit into a string.
XMLHierarchyRequestError	Raised when an attempt is made to insert a node where the node type is not allowed.
XMLIndexSizeError	Raised when an index or size parameter to a method is negative or exceeds the allowed values.
XMLInuseAttributeError	Raised when an attempt is made to insert an attribute node that is already present elsewhere in the document.
XMLInvalidAccessError	Raised if a parameter or an operation is not supported on the underlying object.
XMLInvalidCharacterError	Raised when a string parameter contains a character that is not permitted in the context it's being used in by the XML 1.0 recommendation.
XMLInvalidModificationError	Raised when an attempt is made to modify the type of a node.
XMLInvalidStateError	Raised when an attempt is made to use an object that is not defined or is no longer usable.
XMLNamespaceError	Raised when an attempt is made to change any object in a way that is not permitted with regard to the Namespaces in XML recommendation.
XMLNotFoundError	Raised when a node does not exist in the referenced context.
XMLNotSupportedError	Raised when the implementation does not support the requested type of object or operation.
XMLNoDataAllowedError	Raised if data is specified for a node which does not support data.
XMLNoDataAllowedError	Raised on attempts to modify an object where modifications are not allowed (such as for read-only nodes).
XMLSyntaxError	Raised when an invalid or illegal string is specified.
XMLWrongDocumentError	Raised when a node is inserted in a different document than it currently belongs to, and the implementation does not support migrating the node from one document to the other.

For example:

```
Set XML=New XMLDocument
XML.Parse("<xml a1='1' a2='2'><a>A</a><b>B</b><c>C</c></xml>")

For Each Element In XML.Elements
   Print Element.Name
Next

Print XML.Search("b").Length

Set Attribute=XML.GetAttributeNode("a1")
Print Attribute.Name & " is ” & Attribute.Value

For Each Attribute In XML.Attributes
   Print Attribute.Name
Next
```

```
Print XML.Compose
```

2.2.3 JSON support

The JSON support includes such functions used for working with JSON as ToJSON and AsJSON.

Function	Description

ToJSON	Accept double, integer, string, array or dictionary as input and convert them to the string in the JSON format. For example:
	`MyArray=Array(1, 2, 3)`
	`Print ToJSON(MyArray)`

AsJSON	Accept string as input parse and return appropriate data. For example:
	`MyArray=AsJSON("[1, 2, 3]")`
	`For Each Item In MyArray`
	`    Print Item`
	`Next`

2.2.5 Network connection support

The network connection support includes such objects used for working with the network connection as Connection and Proxy.

Setting network connection

```
Set MyConnection=New Connection
MyConnection.Open("http://site.net/file")
MyConnection.Encoding="utf-8"
Print MyConnection.Read
MyConnection.Close
```

2.2.5.1 Connection class

Connection is a VScript built-in class representing a class used for opening the network connection with a server and reading and writing the data from it.

Connection class has the following properties:

Property	Description
Encoding	Set the character encoding for receiving and sending the data.
Proxy	Set the list of proxies to use. The value of this property must represent an object of Proxy class.
IsConnected	Return the logical value, which indicates the current state of a network connection. Read-only.

Connection class has the following methods:

Method	Description
Open(URL)	Open the network connection to the given URL. Only HTTP protocol is supported. URL - the URL to which the network connection is opening. Here the full address of the resource must be written.

Method	Description
Read()	Read the data. If the character encoding has been set (in the Encoding property), the result will be returned as a String; otherwise - as a Binary subtype.
Write(Data)	Write the data. If the character encoding has been set (in the Encoding property), a string must be sent as an attribute; otherwise - as a Binary subtype.
Close()	Close the network connection and the current session. The connection can be opened again later.

2.2.5.2 Proxy class

Proxy is a VScript built-in class representing a class used together with the Connection class, which represents a list of proxy. It allows you to create a proxy list for different protocols.

Setting new proxy

```
Set MyProxy=New Proxy
MyProxy("HTTP")="192.168.1.1"
```

3 Plugins development

Each ProSuite application has its own set of classes and functions available for using in macros for plugins. The classes are different in the range of use, and according to it can be grouped into the common classes for all plugins and the classes specific for every ProSuite application.

The description of these classes is also structured in accordance with these two class types.

3.1 Common functions

There are some functions which can be used by developers of macros. They are used for debugging primarily. These functions are as follows:

Function	Data type	Description
generate_guid	String	Generate and return the String containing a globally unique identifier. It is used when you want to generate a unique name. guid = generate_guid logger guid
logger("data")	String	Send a String containing some data to the logger. The Strings can be concatenated. Logger is accessible in the logs page of the ProSuite application. To go to logs in the address bar of your browser write the link to the application, then "/logs". logger("Hello, world!")

3.1 Common classes

There are classes accessible from plugins of any ProSuite application. They are called **common classes**. The description of the common classes for plugins you can find in the sections below.

3.1.1 ProAdmin class

ProAdmin is a common class that provides access to the ProAdmin library. Its main functionality is to work with users and groups registered in the ProAdmin application. The methods of the ProAdmin class return instances of the classes described below.

Method	Data type	Description
current_user	User	Return the instance of the User class, which is currently logged in the application, if such instance exists; or return Nothing if there are no currently logged in users. set current_user = ProAdmin.current_user logger current_user.name
set_user(user)	User	Set the user given in the parameter as a current user. user - the instance of the User class.

		set current_user = ProAdmin.current_user
		logger "Current user name - " & current_user.name
		users = ProAdmin.users("test")
		if Ubound(users) > -1 then
		    set user = users(0)
		    ProAdmin.set_user(user)
		    set current_user = ProAdmin.current_user
		    logger "New current user name - " & current_user.name
		else
		    logger "no such user 'test'"
		end if

login(user, password)		Change the user logged in the system. If during logging in an error occurs, the exception (ProAdminEmptyPasswordError, ProAdminLoginError) will be raised. user - the name of the instance of the User class; password - the user’s password.
		ProAdmin.login("test", "test") set current_user = ProAdmin.current_user logger "Current user name - " & current_user.name
users([email [,guid]])	Array	Return the Array consisting of instances of the User class. Searching by ‘email’ and/or ‘guid’ can also be set. If the parameters of searching are not set, all the users registered in ProAdmin will be returned. email - the user's E-mail; guid - the user's globally unique identifier.

		users = ProAdmin.users("root")
		if ubound(users) > -1 then
		    set user = users(0)
		    logger user.name
		else
		    logger "No such user 'root'"
		end if

groups([name [,guid]])	Array	Return the Array consisting of instances of a Group class. Searching by ‘name and/or ‘guid’ can also be set. If the parameters of searching are not set, all the groups registered in ProAdmin will be returned. name - the name of the instance of the Group class; guid - the group's globally unique identifier.

		groups = ProAdmin.groups("Administrators")
		if ubound(groups) > -1 then
		    set group = groups(0)
		    logger group.name
		else
		    logger "No such group 'root'"
		end if

user_in_group(user, group)	Boolean	Return *True* if the instance of the User class belongs to the group, if not - *False*.

		• user - the name of the instance of the User class; group - the group name.
		groups = ProAdmin.groups("Administrators") set admins = groups(0) users = ProAdmin.users("root") set root = users(0) users = ProAdmin.users("test") set test = users(0) logger "Is user 'root' in group 'Administrators': " & ProAdmin.user_in_group(root, admins) logger "Is user 'test' in group 'Administrators': " & ProAdmin.user_in_group(test, admins)
applications	Array	Return the array of the instances of the Application class registered in ProAdmin.

		apps = ProAdmin.applications
		for each app in apps
		    logger "App name: " & app.name
		    logger "IP: " & app.ip
		    logger "GUID: " & app.guid
		next

3.1.1.1 User class

User is a class returned by the methods of ProAdmin class that contains instances with the information about the users. All values here are read-only. Every instance of this class has the following fields:

Field	Data type	Description
guid	String	A String containing the user's globally unique identifier.
name	String	A String containing the user name.
email	String	A String containing the user's E-mail address.
phone	String	A String containing the user's telephone number.
first_name	String	A String containing the user's first name.
last_name	String	A String containing the user's last name.

In order to get the necessary data use the notation like in the example:

```
User.name
```

This class has the following methods:

Method	Data type	Description
get_groups	Array	Return the Array consisting of instances of the Group class to which the user belongs.
		set user = ProAdmin.current_user groups = user.get_groups for each group in groups logger group.name next

3.1.1.2 Group class

Group is a class returned by the methods of ProAdmin class that contains the instances with the information about the groups. All values here are read-only. Every instance of this class has the following parameters:

Field	Data type	Description
guid	String	A String containing the group's globally unique identifier.
name	String	A String containing the group name.

In order to get the necessary data use the notation like in the example:

```
Group.name
```

This class has the following methods:

Method	Data type	Description

get_users	Array	Return the array containing instances of the User class (the users, belonging to this group).
		`groups = ProAdmin.groups( "Administrators" )`
		`set group = groups(0)`
		`users = group.get_users`
		`for each user in users`
		`    logger user.name`
		`next`

3.1.1.3 Application class

Application is a class returned by the methods of ProAdmin class that contains the instances with the information about the application. All values here are read-only. Every instance of this class has the following parameters:

Field	Data type	Description
guid	String	A String containing the application's globally unique identifier.
name	String	A String containing the application name.
ip	String	A String containing the IP address, which refers to the application.

In order to get the necessary data use the notation like in the example:

```
Application.ip
```

3.1.2 SmartCard class

SmartCard is a common class that stores information about the licenses and settings recorded on a smart card. This class has the following method:

Method	Description
get_parameter(key)	Return the system attributes.

3.1.3 System class

System is a common class that provides the general interface for working with the VDOM server. This class has the following methods:

Method	Data type	Description
application_id	String	Return the String containing the identifier of the application. `logger system.application_id`

application_name	String	Return the String containing the name of the application. `logger system.application_name`

application_hosts	Array	Return the Array containing the virtual hosts of the application.
		
		`hosts = system.application_hosts`
		`for each host in hosts`
		`    logger host`
		`next`

server_version	String	Return the String containing the version of VDOM server. `logger system.server_version`
send_email(to_email, from_email, subject, body)		Send an E-mail. to_email - the E-mail address of the receiver of the E-mail; from_email - the E-mail address of the sender of the E-mail; subject - the subject of the E-mail; body - the content of the E-mail. `system.send_email( "to@user.com", "sender@user.com", "Test message", "Vscript testing" )`
http_request(url)		Make a HTTP request (for example, you can send data via GET method to some API). `system.http_request( "url.com/page?params=1" )`
logger		Send a message to the log. Logger is accessible in the logs page of the ProSuite application. To go to logs in the address bar of your browser write the link to the application, then "/logs". `logger "Hello, world!"`
clear_log		Delete all records from log. `system.clear_log`
enable_debug		Enable writing messages to logs. `system.enable_debug`
disable_debug		Disable writing messages to logs. `system.disable_debug`

3.1.4 DBDictionary class

DBDictionary is a common dictionary-like class that represents a key-value storage, which can be used from VScript. All macros from one plugin have one storage. Macros from different plugins do not have access to the data of each other. Here can be stored only simple data types, like Strings, Integers and so on.

Method	Data type	Description

DBDictionary.remove("key")		Remove all the data from the DBDictionary.
		
		dbdictionary("key") = "value"
		dbdictionary.remove("key")
		if isEmpty(dbdictionary("key")) then
		    logger "value removed from DBDictionary"
		else
		    logger "value in dbdictionary"
		end if

DBDictionary("key")	String	Return the String containing the value from the DBDictionary with the given key. If there is no such key, Empty will be returned. dbdictionary("key") = "value" logger dbdictionary("key")
DBDictionary("key")="value"		Put the value to the DBDictionary. dbdictionary("key") = "value" logger dbdictionary("key")

3.1.5 SessionDictionary class

SessionDictionary is a common dictionary-like class accessible only from the button macros. It represents a key-value storage where the data can be stored during the session. All macros from one plugin have one storage. Macros from different plugins do not have access to the data of each other. Here can be stored not only simple but also complex data types.

Method	Data type	Description

remove("key")		Remove all the data from the SessionDictionary.
		
		SessionDictionary("key") = "value"
		SessionDictionary.remove("key")
		if isEmpty(SessionDictionary("key")) then
		    logger "value removed from SessionDictionary"
		else
		    logger "value in SessionDictionary"
		end if

SessionDictionary("key")		Return a simple data type from the SessionDictionary with the given key. If there is no such key, Empty will be returned. SessionDictionary("key") = "value" logger SessionDictionary("key")
SessionDictionary("key")="value"		Put a simple data type to the SessionDictionary. SessionDictionary("key") = "value" logger SessionDictionary("key")
set SessionDictionary("key"		Put a complex data type to the SessionDictionary.

)= object		
		```
		set SessionDictionary( "key" ) = ProAdmin.current_user
		if isEmpty( SessionDictionary( "key" ) ) then
		   logger "value removed from SessionDictionary"
		else
		   logger "value in SessionDictionary"
		   set user = SessionDictionary( "key" )
		   logger user.name
		end if
		```

set value = SessionDictionary("key")		Return a complex data type from the SessionDictionary with the given key. If there is no such key, Empty will be returned.
		```
		set SessionDictionary( "key" ) = ProAdmin.current_user
		if isEmpty( SessionDictionary( "key" ) ) then
		   logger "value removed from SessionDictionary"
		else
		   logger "value in SessionDictionary"
		   set user = SessionDictionary( "key" )
		   logger user.name
		end if
		```

3.1.6 Buffer class

Buffer is a common class that provides an interface for working with files. It uses all standard principles and functions of working with files.

Method	Data type	Description
create	Buffer	Create an instance of the Buffer class; return the instance. `set buffer = Buffer.create`

write(string)	String	Write a string to a file.
		
		string - an object of String subtype.
		```
		set buf = Buffer.create
		buf.write("Some text")
		buf.seek(0)
		logger buf.read
		```

writelines(array)	String	Write several strings to a file.
		
		array - an object of Array subtype, which includes 0 or more objects of String subtype.
		```
		dim lines(4)
		lines(0) = "Some"
		lines(1) = " "
		lines(2) = "test"
		lines(3) = " "
		lines(4) = "Method writelines"

		set buf = Buffer.create
		buf.writelines( lines )
		buf.seek(0)
		logger buf.read
		```

read([size])	String	Read the bytes from a file; return it as a String. size - optional parameter, sets the maximum size of a string in bites. If it is not set, the whole contents of a file will be returned. system.clear_log set buf = Buffer.create buf.write("Some text") buf.seek(0) logger buf.read buf.seek(0) logger buf.read(2)
readline([size])	String	Read a line from a file; return it as a String. size - optional parameter, sets the maximum size of a string in bites. If it is not set, the whole contents of a file will be returned. set buf = Buffer.create buf.write("Some text") buf.seek(0) logger buf.readline
readlines([size])	Array	Read lines from a file; return it as an Array consisting of Strings. size - optional parameter, sets the maximum size of a string in bites. If it is not set, the whole contents of a file will be returned. set buf = Buffer.create buf.write("Some text") buf.seek(0) lines = buf.readlines for each line in lines logger line next
seek(offset, [whence])		Set new value of a pointer. offset - whence - optional parameter, by default equals to 0, can be set to 1 or 2. set buf = Buffer.create buf.write("Some text") logger "current position: " & buf.tell buf.seek(0) logger "current position: " & buf.tell
tell	Integer	Return the current position of a pointer. set buf = Buffer.create buf.write("Some text") logger "current position: " & buf.tell buf.seek(0) logger "current position: " & buf.tell
truncate([size])		Truncate the file's size.

		size - optional parameter, sets the maximum size of a string in bites. If it is set, the file is truncated to that size. The size defaults to the current position. The current file position is not changed. Note that if a specified size exceeds the file's current size, the result is platform-dependent: possibilities include that the file may remain unchanged, increase to the specified size as if zero-filled, or increase to the specified size with undefined new content.
		`set buf = Buffer.create`
		`buf.write("Some text")`
		`buf.truncate(3)`
		`buf.seek(0)`
		`logger buf.read`

close		Close the file. A closed file cannot be read or written any more.
		`set buf = Buffer.create`
		`buf.write("Some text")`
		`buf.close`

3.1.7 URLLib class

URLLib is a common class that provides an interface for processing data from HTTP request; accepts Universal Resource Locators (URLs); GET and POST methods are supported.

Method	Data type	Description

urlopen(url, [params])		Send a request to the URL address specified by the url argument; return an instance of a Buffer class.
		
		url - a String containing a valid URL address;
		
		params - a dictionary containing parameters.
		Example of GET request
		`url = "test.host.local/testpage.php?param1=asdasd&param2=qwerty"`
		`set buf = URLLib.urlopen(url)`
		Example of POST request
		`url = "test.host.local/testpage.php"`
		Sending a file in ASCII format
		`'creating an array consisting of two elements: the first is the file name, second - the instance of the buffer class`
		`dim myfile(1)`
		`myfilename(0) = "Filename.txt"`
		`set myfilebuffer = Buffer.create`
		`myfilebuffer.write("The contents of a file")`
		`set myfile(1) = myfilebuffer`
		
		`'adding the array to parameters`
		`parameters("parameter3") = myfile1`
		`set buf = URLLib.urlopen(url, parameters)`

3.1.8 XML_Dialog

XML_Dialog is a common object that is accessible only from the button macros. It provides an interface for working with dialogs used to enable communication between an application and a user. Dialogs in button macros are used when user triggers macros, and it is necessary to receive some data or inform user about something.

1.3.8.1 Using the XML_Dialog object in macro's source code

There is a standard way of creating XML_Dialog objects for all web applications based on VDOM technology. This way implies creating an XML description of a dialog written in a special way, which is rendered by application as a dialog window. The XML description consists of a number of standard blocks described below.

The way of macro execution is the following: the macro's source code is executed twice:

- first – when the dialog is shown and receives arguments from user;
- second– when the arguments are already received and they are used in further macro execution.

So, to check whether the macro is executed for the first time or not, add to the XML description of a dialog an invisible field of a TextBox tag in the following format:

The step identifier

```
<TextBox id=""step"" visible=""0""></TextBox>"
```

Start the macro's source code from checking on having the "step" argument:

Check on having the arguments

```
arguments = XML_Dialog.get_answer
operation_step = 1
if "step" in arguments then
   operation_step = 2
end if
```

Here, the **operation_step** variable defines what piece of code will be executed further: if the "step" argument has not been received, the **operation_step** equals 1 which means that the macro is executed for the first time; if the "step" argument is received, the **operation_step** equals 2 which means that the macro is executed for the second time.

The XML description of XML_Dialog object must include the macro's identifier, in order to make the system executing the macro with this identifier once again. The identifier is returned by the get_answer method together with the received data when user commits the data to the server. The macro's identifier is stored in the invisible field of a TextBox block in the following format:

The macro's identifier

```
<TextBox id=" macros_id" visible="0"></TextBox>
```

The tags of XML description of the XML_Dialog object are assigned to a variable as concatenated Strings. The get_macros_id method checks the macro's identifier. For example, here is a piece of macro's source code containing an XML_Dialog object:

The XML description of XML_Dialog object

```
xml_header = "<?xml version='1.0' encoding='utf-8'?><VDOMFormContainer><Components>"
xml_footer = "</Components></VDOMFormContainer>"
```

```
xml_macros_id = "<TextBox id=""macros_id"" visible=""0""><Properties><Property name=""defaultValue"">" &
XML_Dialog.get_macros_id & "</Property></Properties></TextBox>"
xml_step = "<TextBox id=""step"" visible=""0""><Properties><Property
name=""defaultValue"">2</Property></Properties></TextBox>"
xml_msg = ""
```

It makes the code more readable when the tags of XML description are split into parts. The whole dialog that is shown when the **step** variable equals 1, looks in the source code as follows:

The XML_Dialog object

```
xml_msg =  xml_header & xml_macros_id & xml_step & xml_footer
XML_Dialog.show_xml_form( xml_msg )
```

1.3.8.2 Methods of the XML_Dialog object

The way of macros execution is implemented thanks to the methods of the XML_Dialog object. These methods are as follows:

Method	Data type	Description

show_xml_form (xmlform)		Show a dialog to user.
		
		xmlform - the dialog window in XML format.
		
		`data_to_xml = "<?xml version=""1.0"" encoding=""utf-8""?>"`
		`data_to_xml = data_to_xml & "<VDOMFormContainer><Components>"`
		`data_to_xml = data_to_xml & "<Heading visible=""1""><Properties><Property name=""text"">"`
		`data_to_xml = data_to_xml & "Setup sender E-mail"`
		`data_to_xml = data_to_xml & "</Property></Properties></Heading>"`
		`xml_dialog.show_xml_form( data_to_xml )`

get_answer	Dictionary	Return the Dictionary containing the parameters of the XML dialog.
		
		`args = xml_dialog.get_answer`
		`if "arg" in args then`
		
		`    logger args("token")`
		`else`
		`    logger "No such key 'arg' "`
		`end if`

get_macros_id	String	Return the String containing the identifier of a macro. `logger xml_dialog.get_macros_id`

1.3.8.3 Elements of the XML_Dialog object

The XML description of an XML_Dialog object may include blocks enclosed into the main VDOMFormContainer block.

1.3.8.3.1 VDOMFormContainer

It is a mandatory block that represents a container which includes the Properties and the Components blocks:

The scheme of XML representation of the dialog

```
<VDOMFormContainer>
   <Properties>
      <Property name="theme">The predefined data that describe the general theme</Property>
   </Properties>
   <Components>
   ...
   </Components>
</VDOMFormContainer>
```

Other blocks are enclosed into the Components block. There are such blocks as described below. You can find some examples of using the elements of the XML_Dialog object below.

1.3.8.3.2 Heading

It is a mandatory block that represents a title of a dialog:

The scheme of XML representation of the dialog

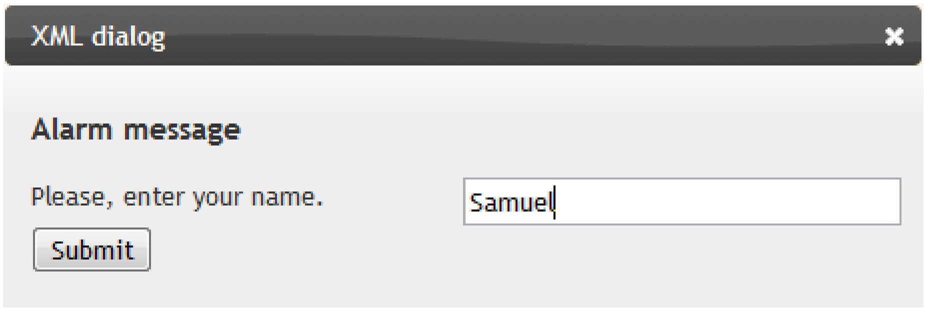
```
<Heading>
   <Properties>
      <Property name="text"><!--[CDATA[The text to show as a title]]--></Property>
      <Property name="subheading"><!--[CDATA[The text to show under the title]]--></Property>
   </Properties>
</Heading>
```

1.3.8.3.3 TextBox

It is a mandatory block that includes the macro's identifier and the field for the data input:

The scheme of XML representation of the dialog

```
<TextBox id=" macros_id " visible="0">
   <Properties>
      <Property name="label"><!--[CDATA[The label of the textbox]]--></Property>
      <Property name="size">The size of a textbox</Property>
      <Property name="maxSize">The maximum number of characters a user can enter</Property>
      <Property name="defaultValue"><!--[CDATA[The value set as a default one]]--></Property>
   </Properties>
</TextBox>
```



XML representation of the dialog

```
<VDOMFormContainer>
   <Components>

<Heading>
   <Properties>
      <Property name="text"><![CDATA[Alarm message]]></Property>
   </Properties>
```

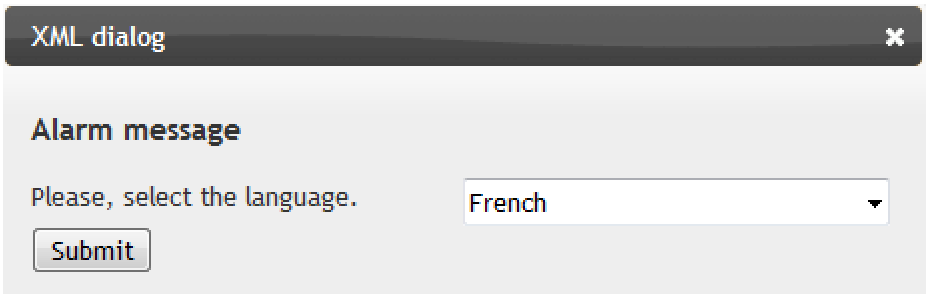
```
</Heading>

<TextBox id="id-for-textbox" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Please, enter your name.]]--></Property>
      <Property name="defaultValue"><!--[CDATA[]]-->Samuel</Property>
   </Properties>
</TextBox>

   </Components>
</VDOMFormContainer>
```

1.3.8.3.4 Dropdown

It is an optional block that includes an element that allows user to choose one value from a list:

The scheme of XML representation of the dialog

```
<DropDown id=" dropdown_id" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[The label of the dropdown]]--></Property>
      <Property name="options">
         <option id="1">Value 1</option>
         <option id="2">Value 2</option>
         <option id="3">Value 3</option>
      </Property>
   </Properties>
</DropDown>
```


XML dialog

Alarm message

Please, select the language. French

XML representation of the dialog

```
<VDOMFormContainer>
   <Components>

<Heading>
   <Properties>
      <Property name="text"><![CDATA[Alarm message]]></Property>
   </Properties>
</Heading>

<DropDown id="id-for-dropdown" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Please, select the language.]]--></Property>
      <Property name="options">
         <option id="1">French</option>
         <option id="2">English</option>
         <option id="3">Russian</option>
      </Property>
   </Properties>
</DropDown>
```

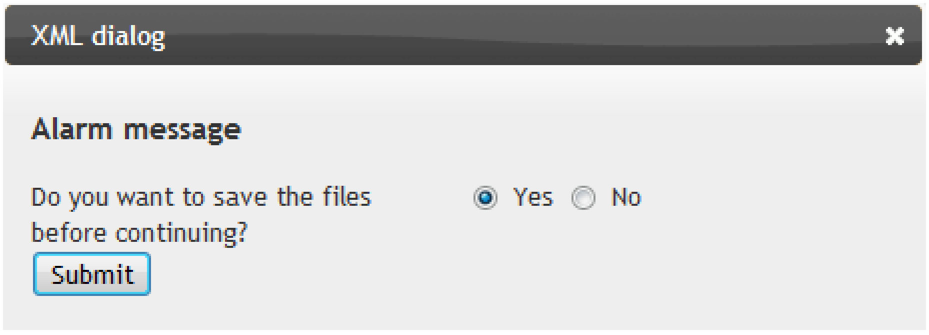
```
    </Components>
</VDOMFormContainer>
```

1.3.8.3.5 RadioButton

It is an optional block that includes an element that allows user to choose only one of a predefined set of options:

The scheme of XML representation of the dialog

```
<RadioButton id="radiobitton_id" visible="1" selected="The option id set as the default one">
    <Properties>
        <Property name="label"><!--[CDATA[The label of the radiobutton]]--></Property>
        <Property name="options">
            <option id="1">Value 1</option>
            <option id="2">Value 2</option>
            <option id="3">Value 3</option>
        </Property>
    </Properties>
</RadioButton>
```



Alarm message

Do you want to save the files before continuing? [✓ Yes] No

XML representation of the dialog

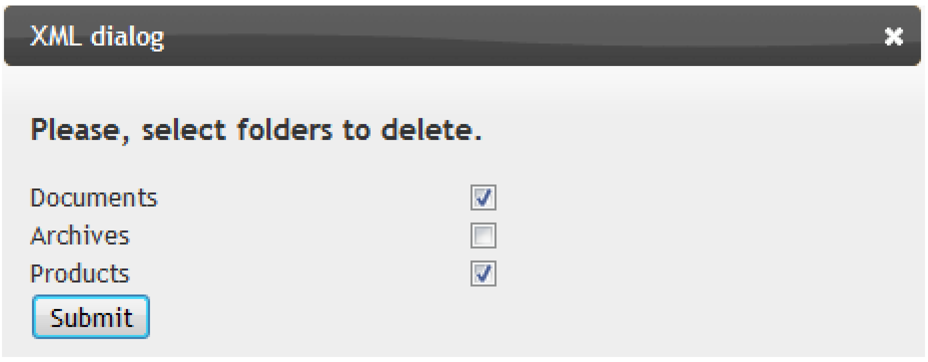
```
<VDOMFormContainer>
    <Components>

<Heading>
    <Properties>
        <Property name="text"><![CDATA[Alarm message]]></Property>
    </Properties>
</Heading>

<RadioButton id="radio-bitton-id" visible="1" >
    <Properties>
        <Property name="label"><!--[CDATA[Do you want to save the files before continuing?]]--></Property>
        <Property name="options">
            <option id="1">Yes</option>
            <option id="2">No</option>
        </Property>
    </Properties>
</RadioButton>

    </Components>
</VDOMFormContainer>
```

1.3.8.3.6 CheckBox

It is an optional block that includes an element that allows user to make multiple selections from a number of options or to have the user answer yes (checked) or no (not checked) on a simple yes/no question:

The scheme of XML representation of the dialog

```
<CheckBox id="checkbox_id" visible="1" selected="true">
   <Properties>
      <Property name="label"><!--[CDATA[The label of the checkbox]]--></Property>
   </Properties>
</CheckBox>
```


XML dialog

Please, select folders to delete.

Documents ✓
Archives
Products ✓

XML representation of the dialog

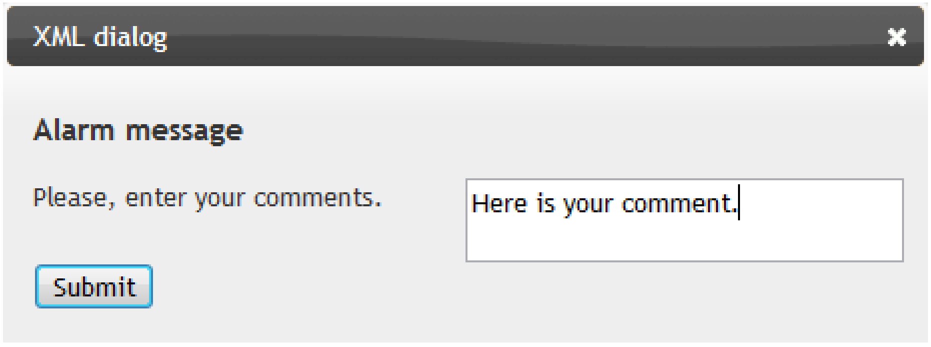
```
<VDOMFormContainer>
   <Components>

<Heading>
   <Properties>
      <Property name="text"><![CDATA[Please, select folders to delete.]]></Property>
   </Properties>
</Heading>

<CheckBox id="checkbox-id1" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Documents]]--></Property>
      <Property name="options">
         <option id="1"></option></Property>
      <Property name="selected"><![CDATA[]]></Property>
   </Properties>
</CheckBox>

<CheckBox id="checkbox-id2" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Archives]]--></Property>
      <Property name="options">
         <option id="2"></option></Property>
      <Property name="selected"><![CDATA[]]></Property>
   </Properties>
</CheckBox>

<CheckBox id="checkbox-id3" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Products]]--></Property>
      <Property name="options">
          <option id="3"></option></Property>
      <Property name="selected"><![CDATA[]]></Property>
   </Properties>
</CheckBox>

   </Components>
</VDOMFormContainer>
```

1.3.8.3.7 TextArea

It is an optional block that includes a field for data input:

The scheme of XML representation of the dialog

```
<TextArea id="textarea_id" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[The label of the textarea]]--></Property>
      <Property name="defaultValue"><!--[CDATA[The value set as a default one]]--></Property>
      <Property name="height">The height of a textarea in pixels</Property>
      <Property name="width">The width of a textarea in pixels</Property>
   </Properties>
</TextArea>
```



XML representation of the dialog

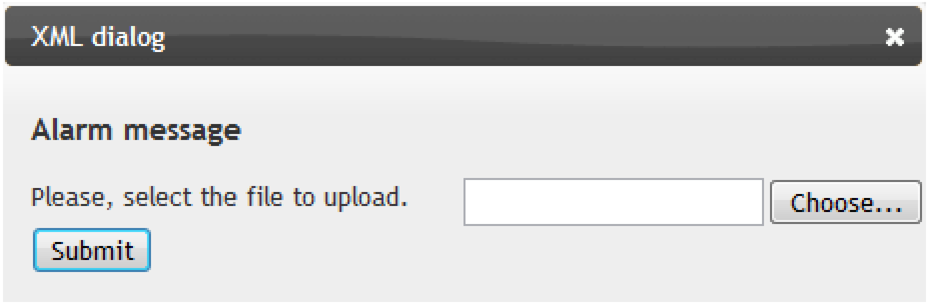
```
<VDOMFormContainer>
   <Components>

<Heading>
   <Properties>
      <Property name="text"><![CDATA[Alarm message]]></Property>
   </Properties>
</Heading>

<TextArea id="textarea-id" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Please, enter your comments.]]--></Property>
      <Property name="defaultValue"><!--[CDATA[Here is your comment.]]--></Property>
   </Properties>
</TextArea>

   </Components>
</VDOMFormContainer>
```

1.3.8.3.8 Upload

It is an optional block that used for uploading files from a local file system to the server:

The scheme of XML representation of the dialog

```
<Upload id=" upload_id" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[The label of the upload]]--></Property>
   </Properties>

</Upload>
```



XML representation of the dialog

```
<VDOMFormContainer>
   <Components>

<Heading>
   <Properties>
      <Property name="text"><![CDATA[Alarm message]]></Property>
   </Properties>
</Heading>

<Upload id="id-for-upload" visible="1">
   <Properties>
      <Property name="label"><!--[CDATA[Please, select the file to upload.]]--></Property>
   </Properties>
</Upload>

   </Components>
</VDOMFormContainer>
```

3.2 Specific classes for plugins

The classes accessible only from a definite ProSuite application are called **specific classes**. The description of the specific classes is structured according to the application where they can be used. You can find it in the sections below.

3.2.1 ProShare

3.2.1.1 Pages where plugins are available

The plugins creating is available on both Files and Smart folders pages of the ProShare application. From this pages plugins can accumulate the information about the selected files and folders and use it in the macro's source code.

3.2.1.2 ProShare global classes

3.2.1.2.1 ProShare class

ProShare is a global class that provides an interface to the ProShare application. It has the following methods:

Method	Data type	Description

get_by_path("path")	Folder/File	Return the instance of the class stored in the specified path as instance of the File or the Folder class. If there is no such instance, Nothing will be returned. For example:
		`set file = ProShare.get_by_path("/temp.file")`
		`logger file.name`

root	Folder	Return the root directory as instance of the Folder class. For example:

		`set root_folder = ProShare.root`
		`logger root_folder.path`

3.2.1.2.2 Page_Status class

Page_Status is a global class that is used for working with the session of the user who pressed the button. This class is accessed only from the button macros. The current status of the application is passed in this class.

Method	Data type	Description
selected_nodes	Array	Return the Array containing the instances of File class, which have been selected in the tree on the main page of the ProShare application.

		`nodes = page_status.selected_nodes`
		`for each node in nodes`
		`    logger node.name`
		`    logger node.path`
		`next`

selected_smartfolders	Array	Return an Array containing the instances of SmartFolder class, which have been selected in the tree on the Smart folder page of the ProShare application.

		`folders = page_status.selected_smartfolders`
		`for each folder in folders`
		`    logger node.name`
		`next`

current_dir	String	Return a String containing the current directory.
		`logger page_status.current_dir`

3.2.1.3 ProShare classes

The instances of classes of the ProShare application are returned by the global methods of the application. The classes of the ProShare application are described below.

3.2.1.3.1 File class

File is a class that provides an interface for working with files. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description
guid	String	A String containing the globally unique identifier of the file.

		`set file = ProShare.get_by_path( "/ad.txt" )`
		`logger file.guid`

name	String	A String containing the name of the file.

		`set file = ProShare.get_by_path( "/ad.txt" )`
		`logger file.name`

upload_date	String	A String containing the date of loading the file to the server (in the following format: "%d/%m/%y %H:%M:%S").

		set file = ProShare.get_by_path("/ad.txt") logger file.upload_date
modification_date	String	A String containing the date of the last file modification (in the following format: "%d/%m/%y %H:%M:%S"). set file = ProShare.get_by_path("/ad.txt") logger file.modification_date
mimetype	String	A String containing the MIME type of the file. set file = ProShare.get_by_path("/ad.txt") logger file.mimetype
serialize_string	String	A String containing the serialized data used in search. set file = ProShare.get_by_path("/ad.txt") logger file.serialize_string
size	Integer	An Integer containing the size of a file in bytes. set file = ProShare.get_by_path("/ad.txt") logger file.size
path	String	A String containing the path to the file. set file = ProShare.get_by_path("/ad.txt") logger file.path

It has the following methods:

Method	Data type	Description
root	Folder	Return the root directory where the file is stored as the instance of a Folder class. set file = ProShare.get_by_path("/ad.txt") set root = file.root logger root.path
move(path)		Move the file to the directory specified by the argument. path - the path where to move the file. set file = ProShare.get_by_path("/ad.txt") file.move("/test") set file = ProShare.get_by_path("/test/ad.txt") logger file.path
copy(path)		Copy the file to the directory specified by the argument. path - the path where to copy the file. set file = ProShare.get_by_path("/ad.txt") file.copy("/test") set file = ProShare.get_by_path("/test/ad.txt")

		logger file.path

delete		Delete the file.
		
		set file = ProShare.get_by_path("/test/ad.txt")
		file.delete
		set file = ProShare.get_by_path("/test/ad.txt")
		if isNothing(file) then
		    logger "file deleted"
		else
		    logger "file exists"
		end if

parent	Folder	Return the root directory where the file is stored as the instance of a Folder class. If the file is stored in the root directory, Nothing will be returned. set file = ProShare.get_by_path("/test/ad.txt") set parent = file.parent logger parent.path logger parent.name
open	Buffer	Return an instance of the Buffer class containing a file. The AccessDeniedError exception is possible to be raised. set file = ProShare.get_by_path("/ad.txt") set buf = file.open logger buf.read
write(handler)		Write data to the file. The contents of a file stored in it before calling this method will be lost. The AccessDeniedError exception is possible to be raised. handler - an instance of the buffer class. set file = ProShare.get_by_path("/ad.txt") set buf = buffer.create buf.write("some text") buf.seek(0) file.write(buf)
get_by_guid(guid)	File	Return an instance of the File or Folder class by its globally unique identifier. If there is no such instance, Nothing will be returned. set file = ProShare.node.get_by_guid("guid") if isNothing(file) then logger "File doesn't exists" else logger file.name end if

3.2.1.3.2 Folder class

Folder is a class that provides an interface for working with folders. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description
guid	String	A String containing the globally unique identifier of the folder.

		set folder = ProShare.get_by_path("/ad") logger folder.guid
name	String	A String containing the name of the folder. set folder = ProShare.get_by_path("/ad") logger folder.name
upload_date	String	A String containing the date of creating the folder (in the following format: "%d/%m/%y %H:%M:%S"). set folder = ProShare.get_by_path("/ad") logger folder.upload_date
modification_date	String	A String containing the date of the last folder modification (in the following format: "%d/%m/%y %H:%M:%S"). set folder = ProShare.get_by_path("/ad") logger folder.modification_date
mimetype	String	A String containing the MIME type of the file (always returns the instance of a folder). set folder = ProShare.get_by_path("/ad") logger folder.mimetype
path	String	A String containing the path to the folder. set folder = ProShare.get_by_path("/ad") logger folder.path

It has the following methods:

Method	Data type	Description
mkdir(new_folder_name)	Folder	Create new folder in the current directory; return the instance of the Folder class, which provides an interface to the folder. If during the creation an error occurs, an exception will be raised. new_folder_name - a String containing the name of the folder. set root = ProShare.root root.mkdir("new folder")
child_nodes	Array	Return the Array containing the instances of classes File and Folder stored in the folder. set root = ProShare.root nodes = root.child_nodes for each node in nodes logger node.name next
root	Folder	Return the root directory.
move(path)		Move the folder to the directory specified by the argument. path - the path where to move the folder.

		```
		set folder = ProShare.get_by_path( "/ad" )
		folder.move( "/test" )
		set folder = ProShare.get_by_path( "/test/ad" )
		logger folder.path
		```
copy(path)		Copy the folder to the directory specified by the argument. path - the path where to copy the folder.
		```
		set folder = ProShare.get_by_path( "/ad" )
		folder.copy( "/test" )
		set folder = ProShare.get_by_path( "/test/ad" )
		logger folder.path
		```
delete		Delete the folder.
		```
		set folder = ProShare.get_by_path( "/ad" )
		folder.delete
		```
parent	Folder	Return the instance of a Folder class. If the file is stored in the root directory, Nothing will be returned.
		```
		set folder = ProShare.get_by_path( "/ad" )
		set parent = folder.parent
		logger parent.path
		```
create_file(name, handler)		Create new file in a folder. name -a name of a file; handler - an instance of the buffer class. The AccessDeniedError, FileAlreadyExistsError, IllegalCharactersInNameError, LongNameError exceptions are possible to be raised.
		```
		set folder = ProShare.get_by_path( "/test" )
		set buf = buffer.create
		buf.write( "some text" )
		buf.seek(0)
		folder.create_file( "new_file", buf )
		```
get_by_guid(guid)	Folder/File	Return an instance of the File or Folder class by its globally unique identifier. If there is no such instance, Nothing will be returned.
		```
		set folder = ProShare.node.get_by_guid( "guid" )
		if isNothing( folder ) then
		   logger "Folder doesn't exists"
		else
		   logger folder.name
		end if
		```

The exceptions which can be raised by the classes **File** and **Folder**:

Exception	Description
DestinationNotFoundError	Destination folder not found.
UnknownDestinationTypeError	Destination type is not supported.
DirectoryAlreadyExistsError	Destination type is not supported.

AccessDeniedError	You have no permissions to perform this action.
NodeLockedError	Node locked.
FileAlreadyExistsError	Destination already contains file with same name.
FolderAlreadyExistsError	Destination already contains folder with same name.
IllegalCharactersInNameError	Name can't contain any of following characters: *?:\\|/"<>.
LongNameError	Name length is over than 100.

3.2.1.3.3 SmartFolder class

SmartFolder is a class that provides an interface for working with smart folders. All the parameters of this class are read-only. Every instance has the following fields:

Method	Data type	Description
guid	String	A String containing the globally unique identifier of the smart folder. set smartfolder = ProShare.SmartFolder.get_by_name("test") logger smartfolder.guid
name	String	A String containing the name of the smart folder. Can be changed. After changing the name of a SmartFolder call the save method. set smartfolder = ProShare.SmartFolder.get_by_name("test") logger smartfolder.name
objects_count	String	A String containing the number of elements in the smart folder. set smartfolder = ProShare.SmartFolder.get_by_name("test") logger smartfolder.objects_count
public_link	String	A String containing the public link to the smart folder. set smartfolder = ProShare.SmartFolder.get_by_name("test") logger smartfolder.public_link
content	Dictionary	A Dictionary containing the list of folders stored in the current smart folder. It is returned as a dictionary-like object, where key is a globally unique identifier of the folder and value is a name of the folder which the folder had when was added to the smart folder. set smartfolder = ProShare.SmartFolder.get_by_name("test") content = smartfolder.content for each guid in content logger guid logger content(guid) next

It has the following methods:

Method	Data type	Description
meta_value(key)	Array	Return the Array containing the metadata. set smartfolder = ProShare.SmartFolder.get_by_name("test") meta_array = smartfolder.meta_value("watcher_emails") for each email in meta_array logger email next

add_meta_value(key, value)		Add data to metadata.
		
		`set smartfolder = ProShare.SmartFolder.get_by_name( "test" )`
		`smartfolder .add_meta_value( "watcher_emails", "sdasd@mail.mail" )`
		`meta_array = smartfolder.meta_value( "watcher_emails" )`
		`for each email in meta_array`
		`    logger email`
		`next`

remove_meta_value(key, value)		Remove data from metadata.

add_folder(folder_guid, label)		Add a folder to the smart folder.
		
		folder_guid - the globally unique identifier of the instance of the Folder class;
		
		label - (optional) the name of the folder for displaying. If it is not set the name of the instance of the Folder class will be used as its label.
		
		`set smartfolder = ProShare.SmartFolder.get_by_name( "test" )`
		`set folder = ProShare.get_by_path("/1")`
		`smartfolder.add_folder( folder.guid, "new_folder" )`

remove_folder(folder_guid)		Remove a folder from the smart folder.
		
		folder_guid - the globally unique identifier of the instance of the Folder class.
		
		`set folder = ProShare.get_by_path("/1")`
		`smartfolder.remove_folder( folder.guid )`

get_by_guid(guid)	SmartFolder/Nothing	Return an instance of the SmartFolder class by its globally unique identifier. If such instance does not exist, then Nothing will be returned.
		
		`set smartfolder = ProShare.SmartFolder.get_by_guid( "123-123-123-123" )`
		`if isNothing( smartfolder ) then`
		`    logger "Smartfolder doesn't exist"`
		`else`
		`    logger smartfolder.name`
		`end if`

get_by_name(name)	SmartFolder/Nothing	Return an instance of the SmartFolder class by its name. If such instance does not exist, then Nothing will be returned.
		
		`set smartfolder = ProShare.SmartFolder.get_by_name( "asdasdasd" )`
		`if isNothing( smartfolder ) then`
		`    logger "Smartfolder doesn't exist"`
		`else`
		`    logger smartfolder.name`
		`end if`

get_all	Array	Return the Array containing all instances of the SmartFolder class.
		
		`all_smartfolders = ProShare.SmartFolder.get_all()`
		`for each smartfolder in all_smartfolder`

		`logger smartfolder.name`
		`next`

create(name)	SmartFolder	Create and return new instance of the SmartFolder class.
		
		name - a name of instance of the SmartFolder class.
		
		`set smartfolder = ProShare.SmartFolder.create( "new smart folder" )`
		`logger smartfolder.name`

The exceptions which can be raised by the SmartFolder class:

Exception	Description
SFAlreadyExistsError	Smart Folder with the same name already exists.
MetaAlreadyExistsError	Meta field with the same name already exists.
FolderAlreadyContainedError	Folder already contained.
IllegalCharactersInNameError	Name cannot contain any of the following characters: *?:\\\|/"<>.

3.2.1.4 ProShare events

The listed below events are used in event plugins creating. You can choose any of them on the step of macro creating when selecting its type and features. The selected event will trigger the macro execution.

Event	Description
AddSmartFolder	The event occurs when the smart folder is created. object - the SmartFolder object, which has been added.
EditSmartFolder	The event occurs when the smart folder is edited. object - the SmartFolder object which has been edited.
DeleteSmartFolder	The event occurs when the smart folder is removed. object - the SmartFolder object, which has been removed.
AddFileSmartFolder	The event occurs when a file is added to the smart folder. object - the File object, which has been added.
EditFileSmartFolder	The event occurs when a file is edited in a smart folder. object - the File object, which has been edited; smart_folder - the array of SmartFolder objects, in which a file has been edited.
DeleteFileSmartFolder	The event occurs when a file from is removed from a smart folder. object - the File object, which has been removed; smart_folder - the array of SmartFolder objects, from which a file has been removed.
AddFile	The event occurs when a file is added. object - the File object, which has been added.
EditFile	The event occurs when a file is edited. object - the File object, which has been edited.
DeleteFile	The event occurs when a file is removed. object - the File object, which has been removed.

3.2.2 ProContact

3.2.2.1 Pages where plugins are available

At the moment of creating the current document the plugins creating is available only on the Contacts page of the ProContact application. From this page plugins can accumulate the information about the selected contacts and use it in the macro's source code.

3.2.2.2 ProContact global classes

3.2.2.2.1 Page_Status class

Page_Status is a global class that is used for working with the session of the user who pressed the button. This class is accessed only from the button macros. The current status of the application is passed in this class.

Method	Data type	Description

selected_contacts	Array/Nothing	Return the Array containing the instances of the Contact class, which have been selected on the main page of the ProContact application.
		
		`contacts = page_status.selected_contacts`
		`if ubound( contacts ) > -1 then`
		`    selected = ubound( contacts ) + 1`
		`    logger selected & " contacts was selected"`
		`else`
		`    logger "Nothing selected"`
		`end if`

selected_contactlist	ContactList/Nothing	Return the instance of the ContactList class, which have been selected on the main page of the ProContact application.
		
		`set contactlist = page_status.selected_contactlist`
		`logger contactlist.name`

3.2.2.3 ProContact classes

The instances of classes of the ProContact application are returned by the global methods of the application. The classes of the ProContact application are described below.

3.2.2.3.1 Contact class

Contact is a class that provides an interface for working with contacts. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description

id	String	A String containing the globally unique identifier of the contact.
		
		`contacts = page_status.selected_contacts`
		`set contact = contacts( 0 )`
		`logger contact.id`

prefix_name	String	A String containing the name that goes before the first name of a contact, such as Mr., Mrs. and so on.
		
		`contacts = page_status.selected_contacts`
		`set contact = contacts( 0 )`

		logger contact.prefix_name
suffix_name	String	A String containing the name that goes after the second name of a contact, such as junior, senior and so on. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.suffix_name
first_name	String	A String containing the first name of the contact. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.first_name
middle_name	String	A String containing the middle name of the contact. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.middle_name
last_name	String	A String containing the last name of the contact. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.last_name
birthday	String	A String containing the date of birth of the contact. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.birthday
company	String	A String containing the company, in which the contact works. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.company
position	String	A String containing the position in the company where the contact works. contacts = page_status.selected_contacts set contact = contacts(0) logger contact.position

3.2.2.3.2 ContactList class

ContactList is a class that provides an interface for working with contact lists. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description
guid	String	A String containing the globally unique identifier of the contact list. set contactlist = page_status.selected_contactlist logger contactlist.guid
name	String	A String containing the name of the contact list.

		set contactlist = page_status.selected_contactlist logger contactlist.name
color	String	A String containing the color of the contact list. set contactlist = page_status.selected_contactlist logger contactlist.color

3.2.2.4 ProContact events

The listed below events are used in event plugins creating. You can choose any of them on the step of macro creating when selecting its type and features. The selected event will trigger the macro execution.

Event	Description
AddContact	The event occurs when the contact is created. object - the Contact object, which has been created.
EditContact	The event occurs when the contact is edited. object - the Contact object, which has been edited.
DeleteContact	The event occurs when the contact is removed. object - the Contact object, which has been removed.
AddContactList	The event occurs when the contact list is created. object - the ContactList object, which has been created.
EditContactList	The event occurs when the contact list is edited. object - the ContactList object, which has been edited.
DeleteContactList	The event occurs when the contact list is removed. object - the ContactList object, which has been removed.
AddContactToList	The event occurs when the contact is added to the contact list. object - the ContactList object, to which the contact has been added; contact - the Contact object, which has been added to the contact list.
RemoveContactFromList	The event occurs when the contact is removed from the contact list. object - the ContactList object, from which the contact has been removed; contact - the Contact object, which has been removed from the contact list.

3.2.3 ProPlanning

3.2.3.1 Pages where plugins are available

At the moment of creating the current document the plugins creating is available only on the Planning page of the ProPlanning application. From this page plugins can accumulate the information about the selected calendars and use it in the macro's source code.

3.2.3.2 ProPlanning global classes

3.2.3.2.1 Page_Status class

Page_Status is a global class that is used for working with the session of the user who pressed the button. This class is accessed only from the button macros. The current status of the application is passed in this class.

Method	Data type	Description

selected_calendars	Array	Return the Array containing the instances of the Calendar class, which have been selected on the main page of the ProPlanning application.
		
		calendars = page_status.selected_calendars
		if ubound(calendars) > - 1 then
		    for each calendar in calendars
		        logger calendar.summary
		        logger calendar.guid
		        logger calendar.color
		    next
		else
		    logger "Nothing selected"
		end if

time_interval	Array	Return the Array containing two objects: the first one in the first day of a month; the second - the last day of a month. The dates are passed in the UNIXTIME format in seconds.
		
		time_interval = page_status.time_interval
		startDate = time_interval(0)
		endDate = time_interval(1)
		logger "Start: " & startDate
		logger "End: " & endDate

calendar_view	String	Return the String containing the current view of the calendar: “month", "week", "day". logger page_status.calendar_view

3.2.3.3 ProPlanning classes

The instances of classes of the ProPlanning application are returned by the global methods of the application. The classes of the ProPlanning application are described below.

3.2.3.3.1 Calendar class

Calendar is a class that provides an interface for working with calendars. All the parameters of this class are read-only.

In **event macros** in order to manipulate the instances of the Calendar class you must set the user of by using the function of **ProAdmin** class set_user.

Every instance has the following fields:

Field	Data type	Description
guid	String/Empty	A String containing the globally unique identifier of the calendar or Empty if there is no data. calendars = page_status.selected_calendars set calendar = calendars(0) logger calendar.guid
summary	String/Empty	A String containing the name of the calendar or Empty if there is no data. calendars = page_status.selected_calendars

		`set calendar = calendars(0)`
		`logger calendar.summary`

color	String/Empty	A String containing the color of the calendar or Empty if there is no data.
		
		`calendars = page_status.selected_calendars`
		`set calendar = calendars(0)`
		`logger calendar.color`

rules	Array	An Array containing the values ("o" - owner, "w" - write access right, "r" - read access right) which denote the access rights for user to the calendar.
		
		`calendars = page_status.selected_calendars`
		`set calendar = calendars(0)`
		`for each rule in calendar.rules`
		`logger rule`
		`next`

It has the following methods:

Method	Data type	Description

get_events(start_date, end_date)	Array	Return the list of instances of the Event class, placed between the dates specified by start_date and end_date arguments.
		
		start_date - the starting date in the UNIXTIME format in milliseconds;
		
		end_date - the ending date in the UNIXTIME format in milliseconds.
		
		`calendars = page_status.selected_calendars`
		`timeInterval = page_status.time_interval`
		`startDate = timeInterval(0)`
		`endDate = timeInterval(1)`
		
		`for each calendar in calendars`
		`    events = calendar.get_events( startDate, endDate )`
		`    count = ubound( events ) + 1`
		`    logger "Calendar '" & calendar.summary & "' has " & count & " events"`
		`next`

save		Save the changes.

add_rules_for_user(user, rules)		Set access rights for the user specified by the user argument.
		
		user - an instance of the User class;
		
		rules - an Array containing the values ("o" - owner, "w" - write access right, "r" - read access right) which denote the access rights for user to the calendar.
		
		`set calendar = proplanning.calendar.create`
		`calendar.summary = "Test calendar"`
		`calendar.color = "#00FF00"`
		`calendar.save()`
		
		`set currentUser = ProAdmin.current_user`
		`dim rules`
		`rules = array( "o", "w", "r" )`
		`calendar.add_rules_for_user( currentUser, rules )`

get_by_guid(guid)	Calendar/Nothing	Return an instance of the Calendar class by its globally unique identifier. If such instance does not exist, then Nothing will be returned.
		```
		set calendar = ProPlanning.Calendar.get_by_guid("some-guid")
		if isNothing(calendar)
		then
		logger "Doesn't exists"
		else
		logger calendar.summary
		end if
		```

create	Calendar	Create and return an instance of the Calendar class.

delete	Calendar	Delete an instance of the Calendar class.
		```
		set calendar = proplanning.calendar.create
		calendar.summary = "Test calendar"
		calendar.save()
		guid = calendar.guid
		calendar.delete()
		set find_calendar = ProPlanning.Calendar.get_by_guid( guid )
		if isNothing( find_calendar ) then
		    logger "Calendar with GUID: " & guid & " doesn't exist"
		else
		    logger "Calendar was found"
		end if
		```

3.2.3.3.2 Event class

Event is a class that provides an interface for working with events. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description
guid	String/Empty	A String containing the globally unique identifier of the event or Empty if there is no data.
summary	String/Empty	A String containing the name of the event or Empty if there is no data.
description	String/Empty	A String containing the description of the event or Empty if there is no data.
rrule	String/Empty	A String containing the recurrence rule or Empty if there is no data.
start_date	Date/Empty	A Date containing the starting date in the UNIXTIME format in milliseconds or Empty if there is no data.
end_date	Date/Empty	A Date containing the ending date in the UNIXTIME format in milliseconds or Empty if there is no data.
until_date	Date/Empty	A Date containing the date until the recurrence rule will be repeated or Empty if there is no data.
all_day	Boolean	A Boolean containing a value defining whether the event lasts all day long or not.

calendar_guid	String/Empty	A String containing the globally unique identifier of the calendar to which current event belongs or Empty if there is no data.
		```
		calendars = page_status.selected_calendars
		timeInterval = page_status.time_interval
		startDate = timeInterval(0)
		endDate = timeInterval(1)

		for each calendar in calendars
		    events = calendar.get_events( startDate, endDate )
		    logger "Calendar " & calendar.summary
		    for each ev in events
		        logger "----------------------------------"
		        logger ev.guid
		        logger ev.summary
		```

		```
		        logger ev.description
		        logger ev.rrule
		        logger ev.start_date
		        logger ev.end_date
		        logger ev.until_date
		        logger ev.all_day
		        logger ev.calendar_guid
		    next
		next
		```

It has the following methods:

Method	Data type	Description
save		Save the changes.
create	Calendar	Create and return an instance of the Calendar class.

delete	Event	Delete an instance of the Event class.
		```
		calendars = page_status.selected_calendars
		set calendar = calendars(0)

		set e = ProPlanning.Event.create
		e.start_date = #31.07.2012#
		e.end_date = #31.07.2012#
		e.summary = "Test event"
		e.calendar_guid = calendar.guid
		e.save()
		e.delete()
		```

3.2.3.3.3 Notification class

Notification is a class that provides an interface for working with notifications. All the parameters of this class are read-only. Every instance has the following fields:

Field	Data type	Description
to_user	User	An instance of the User class denoting a user who receives a notification or Nothing if there is no data. Cannot be set together with the to_email field.
sender	User	An instance of the User class denoting a user who sends a notification or Nothing if there is no data.
event	Event	An instance of the Event class denoting an event about which a user is notified or Nothing if there is no data.
to_email	String	A String containing an E-mail address of the user who receives a notification or Nothing if there is no data. Cannot be set together with the to_user field.

It has the following methods:

Method	Data type	Description
create	Notification	Create and return an instance of the Notification class.

add_notification(notification)	Notification	Add a notification.
		
		notification - an instance of the Notification class.
		```
		set notification = proplanning.notification.create
		set notification.sender = proadmin.current_user
		notification.to_email = "mail@mail.mail"

		set event = ProPlanning.Event.create
		event.summary = "New meeting"
		event.start_date = #31.07.2012#
		event.end_date = #31.07.2012#
		```

		event.save()
		
		set notification.event = event
		
		ProPlanning.notification.add_notification(notification)

3.2.3.4 ProPlanning events

The listed below events are used in event plugins creating. You can choose any of them on the step of macro creating when selecting its type and features. The selected event will trigger the macro execution.

Event	Description
AddCalendarEvent	The event occurs when new calendar is created. calendar - the Calendar object, which has been created.
EditCalendarEvent	The event occurs when a calendar is edited. calendar - the Calendar object, which has been edited.
RemoveCalendarEvent	The event occurs when a calendar is removed. calendar - the Calendar object, which has been removed.
AddEventEvent	The event occurs when an event of a calendar is added. event - the Event object, which has been
EditEventEvent	The event occurs when an event of a calendar is edited. event - the Event object, which has been edited.
RemoveEventEvent	The event occurs when an event of a calendar is removed. event - the Event object, which has been removed.

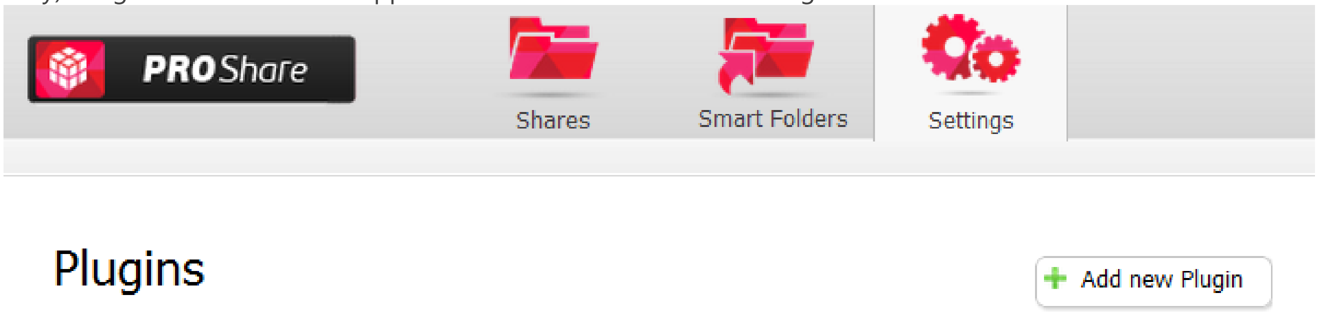
4 Developer's interface

Plugins for ProSuite applications represent a code in XML format, that is why they can be created in any text editor and then imported to the application. Another way of creating plugins, which is more convenient one, is writing them directly in the ProSuite application. It is made possible thanks to the Plugins developing interface implemented in the ProSuite applications.

How to create a plugin, add macros and timers to it, export and import macros you will find in the next sections.

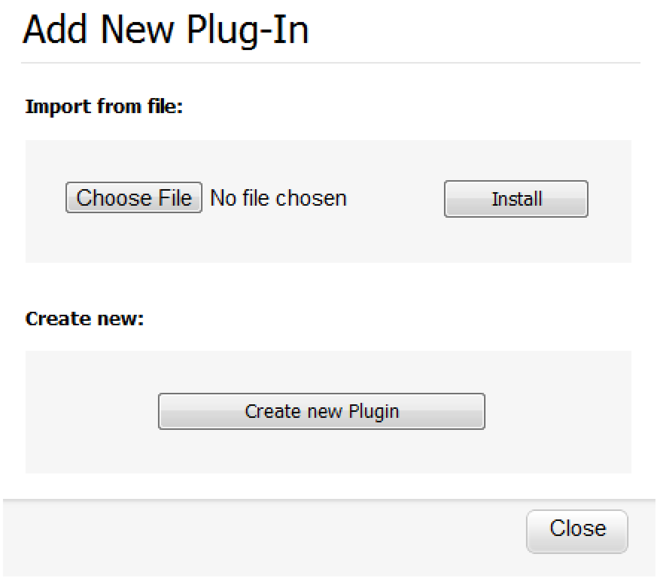
4.1 Plugin creating

The process of plugin creating consists of the following steps: creating the plugin itself, adding to it macros, and timers (if necessary). Plugins for the ProSuite applications can be created in the Plugins section.



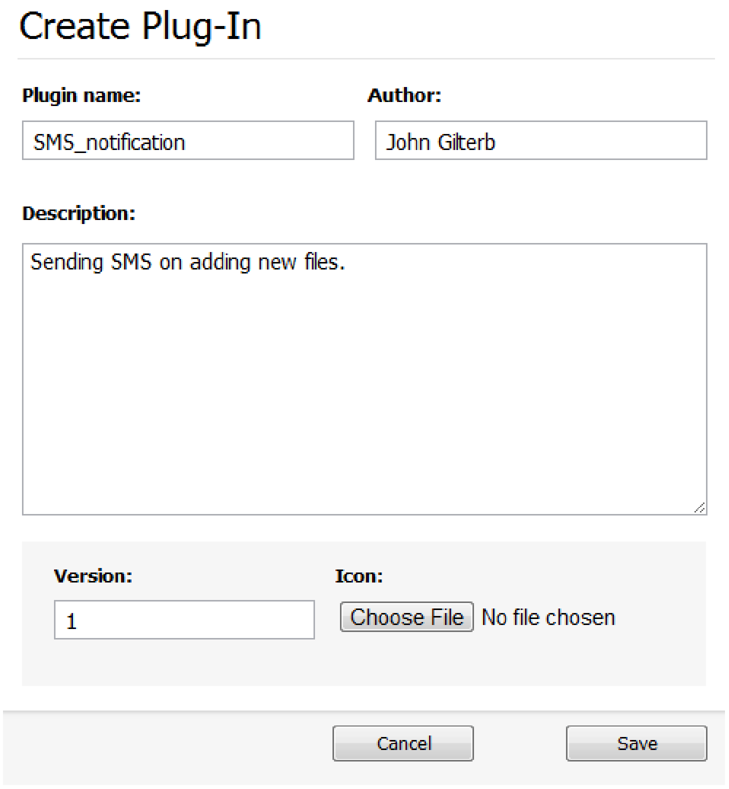
The Plugins section in the ProShare application

After pressing the Add new plugin button a dialog window opens, in which you will be asked to choose between importing and creating plugin. If you want to import a plugin go to the section 1.6 Plug-in importing. If you want to create new plugin, press the Create new plugin button in the dialog window.



A dialog window of adding plugins

Every plugin for the ProSuite applications have the features which will be displayed in the Plugins section of the ProSuite application. So, fill the fields in the opened dialog window: enter the name of the plugin, the author's name, short description, the number of the plugin version and select the icon for it.



Create Plug-In

Plugin name: SMS_notification

Author: John Gilterb

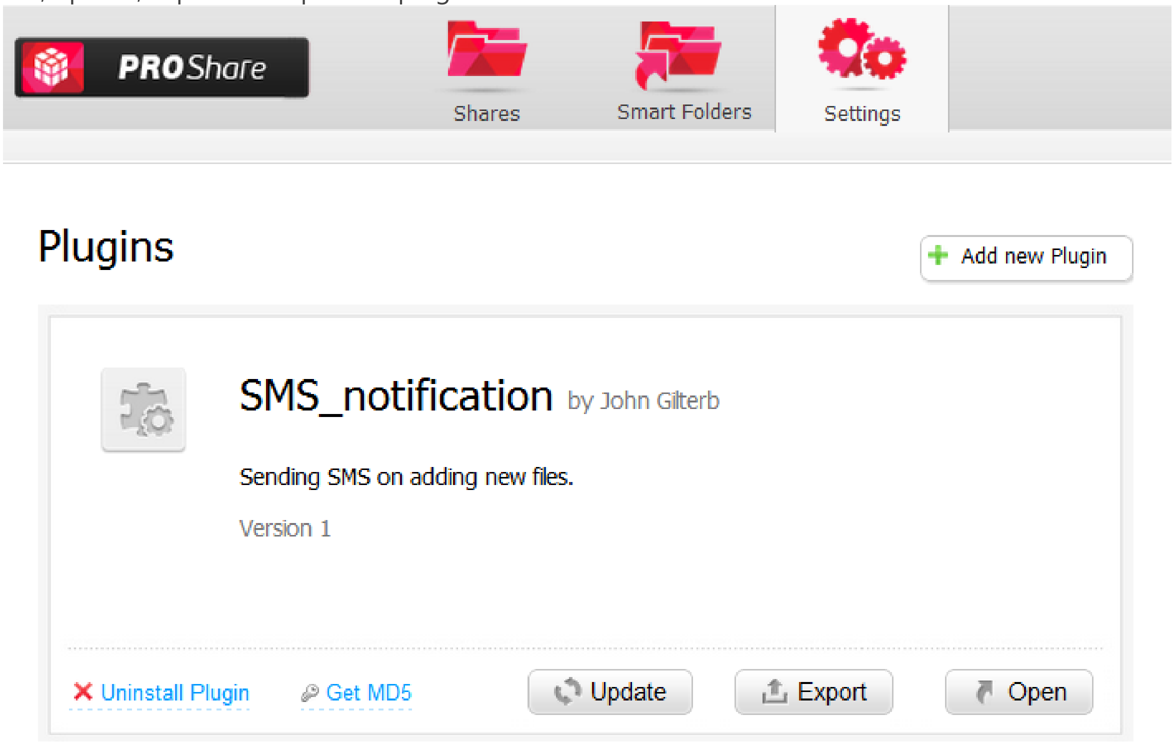
Description: Sending SMS on adding new files.

Version: 1

Icon: Choose File No file chosen

A form for the plugin creating

After saving the changes you will see just created plugin and its features in the Plugins section. Note, that in this section you can delete, update, export and open the plugin.



Plugins

SMS_notification by John Gilterb

Sending SMS on adding new files.

Version 1

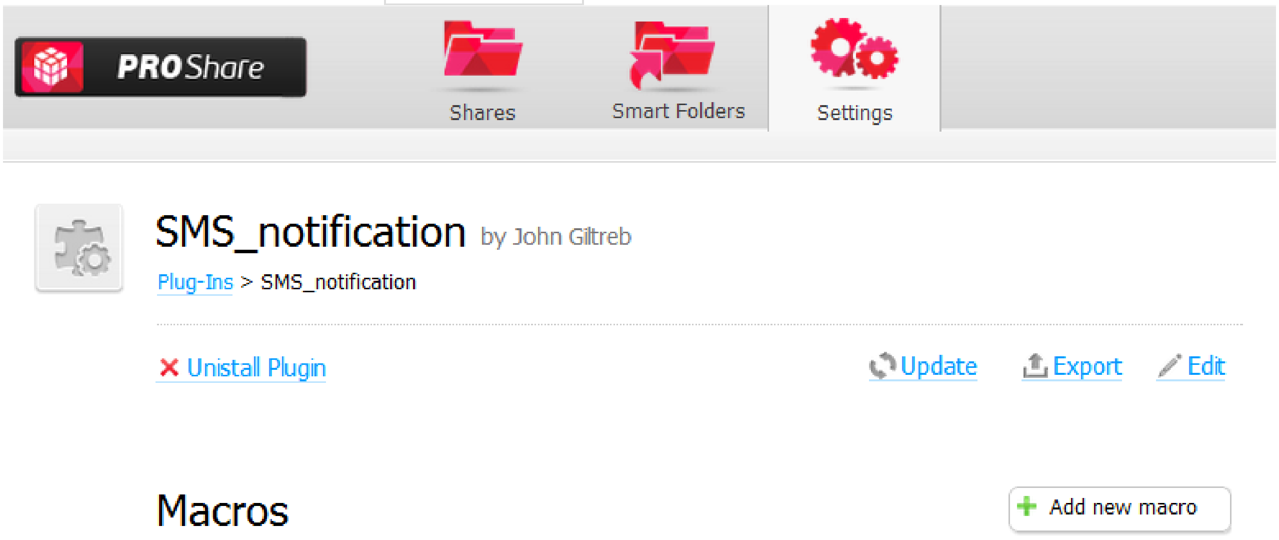
The Plugins section of the ProShare application with just created plugin

The next step of creating a plugin is adding to it macros and timers (if necessary).

4.2 Macros creating

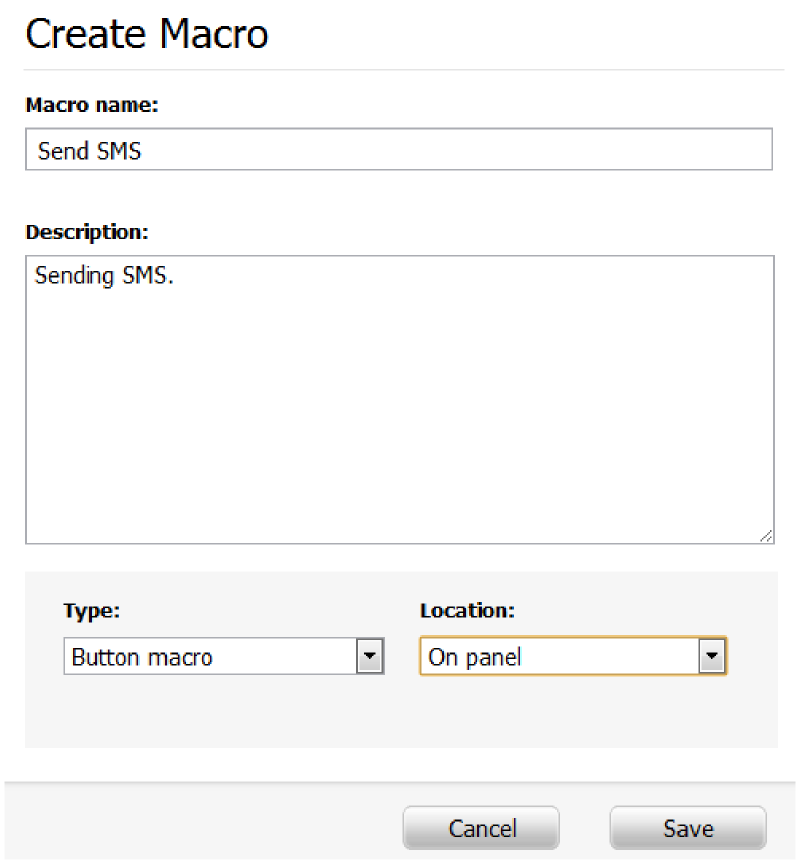
A plugin can consist of one or more macros which execute the main actions of plugin. Macro represents a source code written in the VScript programming language. In order to add a macro to the plugin you must press the Open button to open the plugin.

If you are opening an imported plugin, there can be some already created macros and timers, otherwise the lists of macros and timers will be empty. Press Add new macro to create a macro.



Creating a macro for a plugin in the ProShare application

You will see the dialog window in which you will be asked to fill the fields. Every macro has name and can have a short description. One of the most important things is to specify the necessary macro type (button or event) and its features (location - for button macros, event - for event macros). After filling the fields and selecting the macro type and its features, press the Save button to save the changes and create a macro.


Create Macro

Macro name: Send SMS

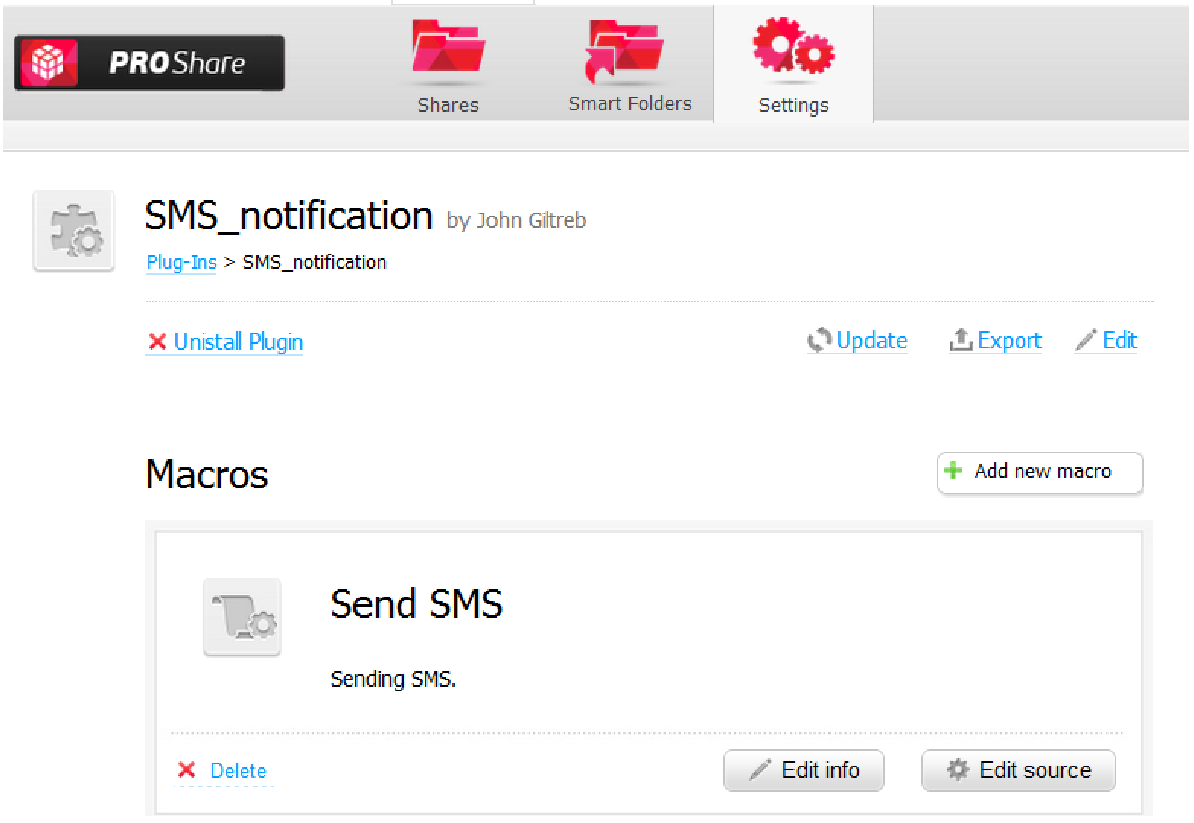
Description: Sending SMS.

Type: Button macro

Location: On panel

A dialog window of a macro creating

After saving the changes the new macro will appear in the plugin. You can modify its description and delete it. In order to write the source code of a macro, press the Edit source button.



SMS_notification by John Giltreb

Plug-Ins > SMS_notification

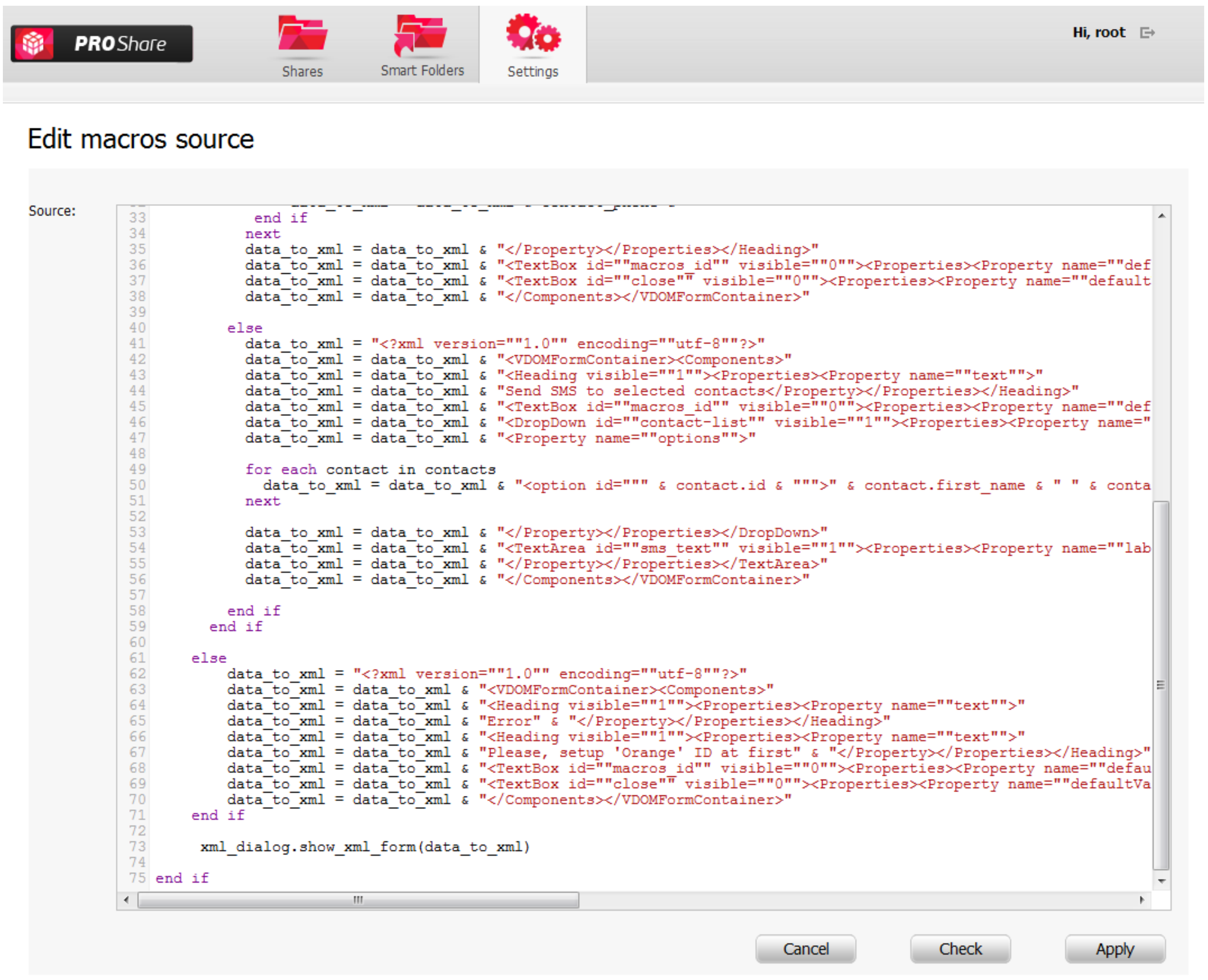
Macros

Send SMS

Sending SMS.

Just created macro of a plugin

The new window will open. In this window you are to write the source code of a macro. You can use the global server objects and the objects of the current ProSuite application in the code. On the whole, macro is that part of a plugin which bears the main functionality and defines what exactly will be executed by the plugin. In order to avoid errors during the plugin execution, press the Check button before saving the macro. The source code of the macro will be checked on errors. Then save the macro by pressing the Save button.



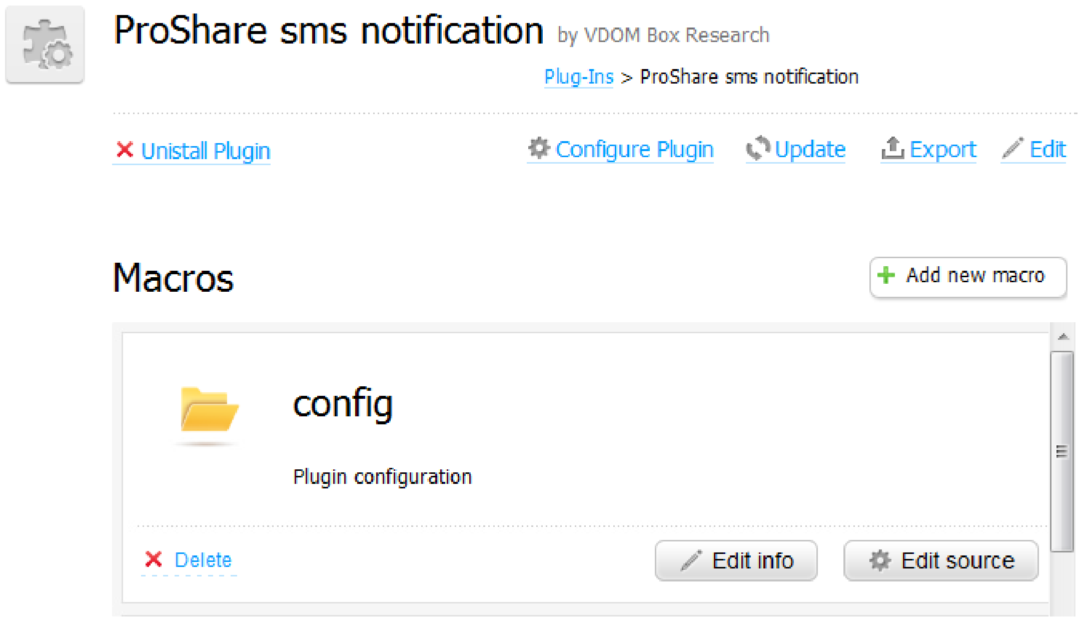
```
           end if
          next
          data_to_xml = data_to_xml & "</Property></Properties></Heading>"
          data_to_xml = data_to_xml & "<TextBox id=""macros_id"" visible=""0""><Properties><Property name=""def
          data_to_xml = data_to_xml & "<TextBox id=""close"" visible=""0""><Properties><Property name=""default
          data_to_xml = data_to_xml & "</Components></VDOMFormContainer>"

        else
          data_to_xml = "<?xml version=""1.0"" encoding=""utf-8""?>"
          data_to_xml = data_to_xml & "<VDOMFormContainer><Components>"
          data_to_xml = data_to_xml & "<Heading visible=""1""><Properties><Property name=""text"">"
          data_to_xml = data_to_xml & "Send SMS to selected contacts</Property></Properties></Heading>"
          data_to_xml = data_to_xml & "<TextBox id=""macros_id"" visible=""0""><Properties><Property name=""def
          data_to_xml = data_to_xml & "<DropDown id=""contact-list"" visible=""1""><Properties><Property name="
          data_to_xml = data_to_xml & "<Property name=""options"">"

          for each contact in contacts
            data_to_xml = data_to_xml & "<option id=""" & contact.id & """>" & contact.first_name & " " & conta
          next

          data_to_xml = data_to_xml & "</Property></Properties></DropDown>"
          data_to_xml = data_to_xml & "<TextArea id=""sms_text"" visible=""1""><Properties><Property name=""lab
          data_to_xml = data_to_xml & "</Property></Properties></TextArea>"
          data_to_xml = data_to_xml & "</Components></VDOMFormContainer>"

        end if
      end if

    else
        data_to_xml = "<?xml version=""1.0"" encoding=""utf-8""?>"
        data_to_xml = data_to_xml & "<VDOMFormContainer><Components>"
        data_to_xml = data_to_xml & "<Heading visible=""1""><Properties><Property name=""text"">"
        data_to_xml = data_to_xml & "Error" & "</Property></Properties></Heading>"
        data_to_xml = data_to_xml & "<Heading visible=""1""><Properties><Property name=""text"">"
        data_to_xml = data_to_xml & "Please, setup 'Orange' ID at first" & "</Property></Properties></Heading>"
        data_to_xml = data_to_xml & "<TextBox id=""macros_id"" visible=""0""><Properties><Property name=""defau
        data_to_xml = data_to_xml & "<TextBox id=""close"" visible=""0""><Properties><Property name=""defaultVa
        data_to_xml = data_to_xml & "</Components></VDOMFormContainer>"
    end if

     xml_dialog.show_xml_form(data_to_xml)

end if
```

The source code of a macro

4.3 Configuration macro

There is one special type of macros called configuration macro. It provides an opportunity of configuring the plugin to which this macro refers. When you add a macro of such type, a Configure plugin button is added to the plugin. On pressing the button the source code of this macro is executed.

To add a configuration macro create a new macro with the name "config". At the top of the window a new button Configure plugin will appear. To open the configuration macro press the Edit source button. This will open the macro, where you can write the source code which will be executed on pressing the Configure Plugin button.



Configuration macro

4.4 Timers creating

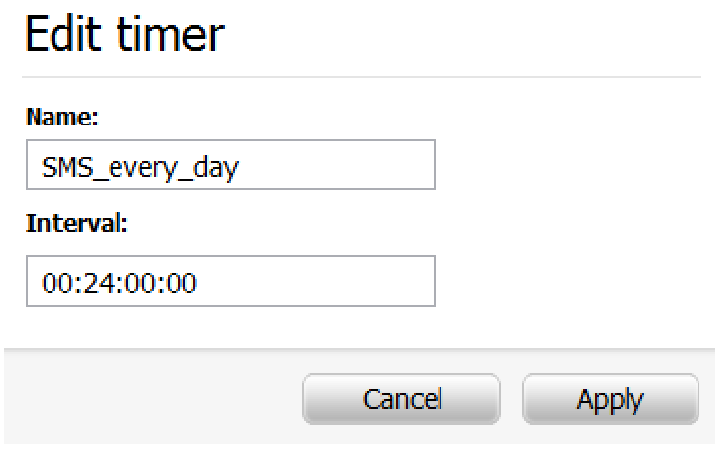
Timers are used in plugins in order to execute the macro's source code on expiry of the specified period of time. For example, if you want some event to occur once a day, you must create a macro and bind a timer to it. The macros which use timers are called the timer macros.

To create a timer open the plugin and press the Add new timer button.

Timers
Add new Timer

Creating a timer for a plugin in the ProShare application

A new window will be opened in which you are to enter a name of a timer and the time interval on expiry of which the event triggering the macro will occur.

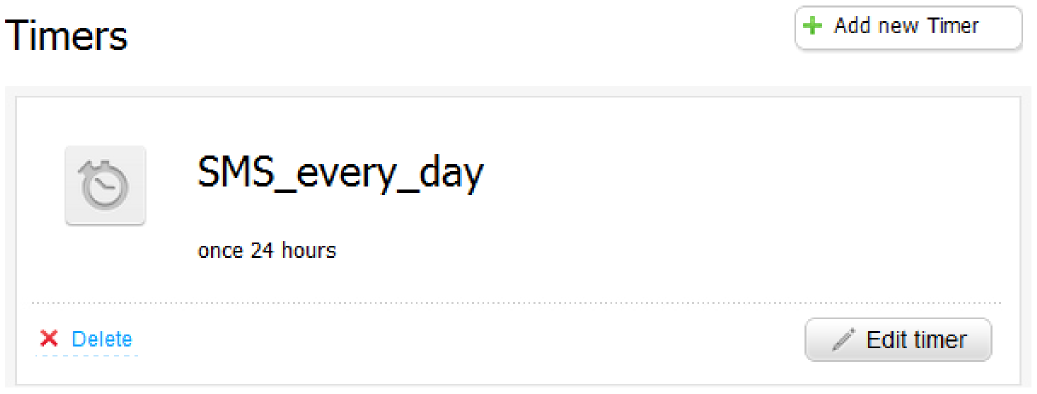

Edit timer

Name: SMS_every_day

Interval: 00:24:00:00

A dialog window of a timer creating

After saving the changes the new timer will appear in the plugin. You can modify its description and delete it.

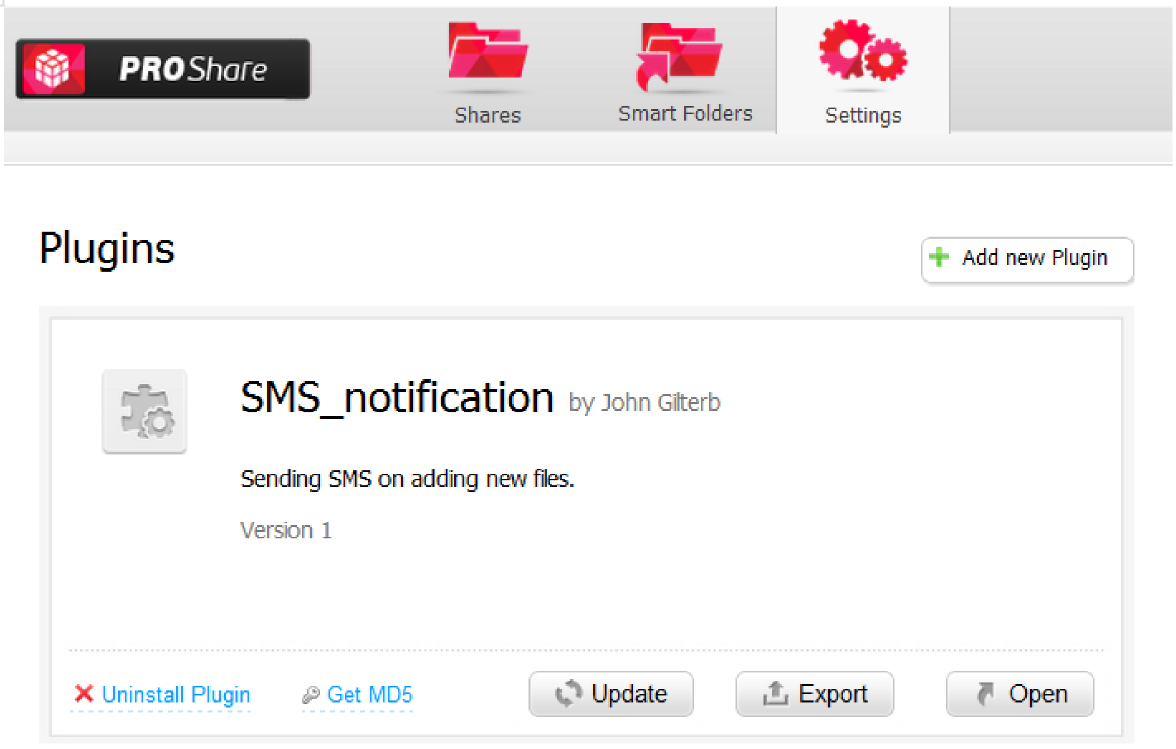


Just created timer of a plugin

4.5 Plugin exporting

Plugin exporting is necessary when you want to install the plugin on the identical ProSuite application located on another server. The plugin will be exported and saved as an XML file, which content you can modify in any text editor.

To export a plugin go to the Plugins section of the ProSuite application, find the plugin which you want to export, press the Export button, specify the location on the computer and save the file.

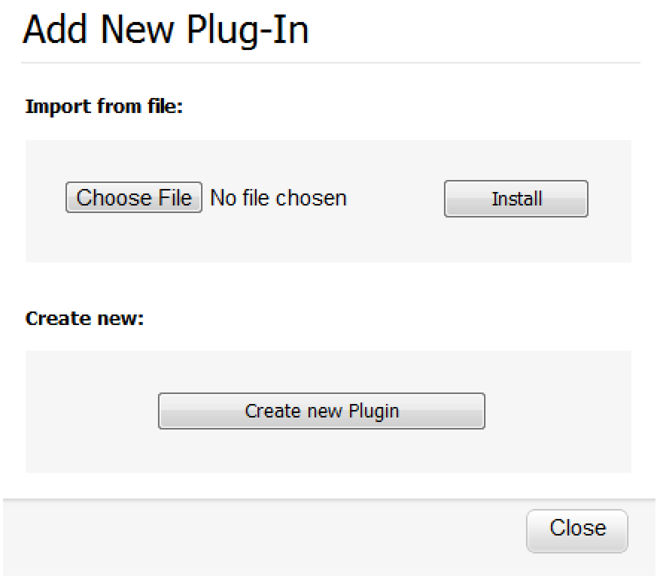


Exporting plugins in the ProShare application

4.6 Plugin importing

If you have an XML file of a plugin for the ProSuite application you can import it. To import a plugin means to install the XML file of a plugin on your ProSuite application.

To import a plugin go to the Plugins section of the ProSuite application and press the Add new plugin button. In the opened window press Choose file and select an XML file of a plugin, then press the Install button. In case the file is damaged or has an incorrect format, you will see the alert message. After finishing the installation the plugin will appear in the Plugins section and you can start using it.



Plugin importing to the ProSuite application

5 ProSuite applications API

Application Programming Interface (API), provided by ProSuite, represents a set of objects and methods collected into a library, which can be used in the macros’ source code when writing plugins for the ProSuite applications. Described below objects and methods of each ProSuite application allow you to integrate two or more applications and exchange the data between them.

The API of each ProSuite application includes the objects accessible from this application and API methods used to perform some operations.

The objects are passed to the methods in JSON format, where the data is structured into a set of key-value pairs. A method returns an Array in JSON format, in which:
the first element - the string that can take two values: "success" (on successful completing the request) or "error" (if an error has occurred);
the second element - the data returned as a result of method execution or an error code.

Execution of each metod may result in "success" and "error". If the request is completed successfully, in most cases the output will contain the data returned by the method. The example of output on success:

```
[ 'success', data ]
```

If during execution some error occurs, the output will contain the error code and the description of it. The example of output on error:

```
[ 'error', error_code, error_description ]
```

5.1 ProAdmin API

5.1.1 ProAdmin objects

5.1.1.1 User

User is an object of the ProAdmin application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user;
password - the password of the user; it is passed only when creating new User object.

5.1.1.2 Application

Application is an object of the ProAdmin application that stores information about the ProSuite applications connected to ProAdmin. It has the following parameters:
guid - the globally unique identifier of the application;
name - the name of the application;
host - the local host of the application.

5.1.2 ProAdmin API methods

Method	Description

create_user(User)	Create new User object; return the globally unique identifier of it. Only administrator can create User objects.
	user - a User object in JSON format.
	`create_user( User ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	For example:
	`create_user( {`
	`    "email"         : "login",`
	`    "first_name"  : "User",`
	`    "last_name"  :  "Userov",`
	`    "password"   :  "some_password"`
	`} )`

delete_users(users_guids)	Delete the User objects; return an empty String if the objects have been deleted successfully. Only administrator can delete User objects.
	users_guids - a list of globally unique identifiers of the User objects.
	`delete_users( users_guids ) : [ 'success' ]`
	For example:
	`delete_users( {`
	`    “users_guids” :  ( [ “6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-`
	`CFCFCFCFCFCF" ]`
	`} )`

login(data)	Login the application; return the User object.
	data - a User object in JSON format.
	`login( data ) : [ 'success', User ]`
	For example:
	`login( {`
	`    "login"    : "login",`
	`    "password" : "password"`
	`} )`

retrieve_applications	Retrieve the registered Application objects; return an Array of them. Only administrator can retrieve the Application objects. `retrieve_applications : [ 'success', [ Application, Application ] ]`
retrieve_users(data)	Retrieve the requested User objects; return an Array of them. data - User objects in JSON format consisting of: users_guids - a list of globally unique identifiers of the User objects. query - a query string. The User objects can be searched by "email". "last_name", "first_name" parameters.

Method	Description

	`retrieve_users( data ) : [ 'success', [ User, User ] ]`
	
	For example:
	
	`retrieve_users( {`
	`    "users_guids" :  [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ],`
	`    "query"        :  "admin"`
	`} )`

update_user(user)	Update the data of the User object; return the globally unique identifiers of it. Only administrator can update the User object.
	user - a User object in JSON format.
	
	`update_user( user ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	
	For example:
	
	`update_user( {`
	`    "guid"          : "123",`
	`    "email"         : "new_login",`
	`    "first_name"  : "new_User",`
	`    "last_name"  :  "new_Userov",`
	`    "password"   :  "new_some_password"`
	`} )`

5.1.3 Errors description

Error code	Error name	Description	Your action
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the objects' structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh does not exist.	Refresh the list of objects.
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.

5.2 ProShare API

5.2.1 ProShare objects

5.2.1.1 User

User is an object of the ProShare application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user.

5.2.1.2 Node

Node is an object of the ProShare application that stores information about files and folders of the application. It has the following parameters:
name - the name of the node; it is returned only when creating new Node object;
path - the absolute path to the node (for example, "/test_folder/test_folder"). When renewing and deleting a Node object, the absolute path including the name of the node must be indicated (for example, "/test/file.doc").

5.2.1.3 File

File is a subobject of the Node object of the ProShare application that stores information about the files of the application. It has the following parameters:
name - the name of the file;
path - the absolute path to the file (for example, "/test_folder/test_folder"); it is passed only by the upload_data method;
b64data - the data of the file stored in base64 format; it is passed only by the upload_data method;
size - the size of the file;
url - the URL of the file;
mimetype - the MIME type of the file. The Folder objects have the “folder” MIME type;
upload_date - the date of uploading the file to the server;
modification_date - the date of modifying the file.

5.2.2 ProShare API methods

Method	Description
create_folder(Node)	Create new folder as an instance of a Node class; return the absolute path to it. Node - a Node object in JSON format.
	create_folder(Node) : ['success', "/folder/new_folder"]
	For example:

	create_folder({
	    "path" : "/folder",
	    "name" : "new_folder"
	})

delete_node(Node)	Delete the Node objects; return an empty String if the objects have been deleted successfully. Node - a Node object in JSON format.
	delete_node(Node) : ['success']
	For example:
	delete_node({

Method	Description
	"path" : "/folder/style.css", })
login(data)	Login the application; return the User object. data - a User object in JSON format. login(data) : ['success', User] For example: login({ "login" : "login", "password" : "password" })
retrieve_folder_data	Retrieve a list of the Node objects, which are stored in a folder; return an Array of the Node objects. Node - a Node object in JSON format. retrieve_folder_data(Node) : ['success', [File, File]] For example: retrieve_folder_data({ "path" : "/folder", })
update_node	Update the name of the Node object; return a path to it. Node - a Node object in JSON format. update_node(Node) : ['success', "/folder/asdsd,xml"] For example: update_node({ "path" : "/folder/style.css", "name" : "new_style.css" })
upload_file	Upload an instance of the File object to the server; return an empty string. File - a File object in JSON format. upload_file(File) : ['success'] For example: upload_file({ "path": "/folder", "b64data": "SGVsbG8hIEl0IGlzIFByb1NoYXJlIHYuMiBBUEk=", "name": "New file" })

5.2.3 Errors description

Error code	Error name	Description	Your action
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the objects' structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh does not exist.	Refresh the list of objects.
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.
8	errFolderAlreadyExist	The Folder object with such name already exists.	
9	errCouldNotDeleteRootDir	The root folder cannot be removed.	
10	errCouldNotRenameRootDir	The root folder cannot be renamed.	
11	errB64DecodeError	An error has occurred when trying to decode Base64.	

5.3 ProContact API

5.3.1 ProContact objects

5.3.1.1 User

User is an object of the ProContact application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user.

5.3.1.2 ContactList

ContactList is an object of the ProContact application that stores information about the contact lists (groups of contacts). It has the following parameters:
guid - the globally unique identifier of the contact list; it is not passed when creating new object of such type;
name - the name of the contact list;
color - (optional) the color of the contact list; it is not returned when creating new ContactList object.

5.3.1.3 Contact

Contact is an object of the ProContact application that stores information about the contacts. It has the following parameters:
id - (optional) the identifier of the contact, it is not passed when creating new Contact object;
prefix_name - the name that goes before the first name of the contact, such as Mr., Mrs. and so on;

suffix_name - the name that goes after the second name of the contact, such as junior, senior and so on;
first_name - the first name of the contact;
middle_name - the middle name of the contact;
last_name - the last name of the contact;
birthday - the date of birth of the contact in UnixTimeStamp format;
company - the company, where the contact works;
position - the position in the company which the contact fills.
picture_url - (optional) the URL address of the image of the contact; it is not passed when creating/renewing the Contact object;
picture_data - (optional) the image of the contact in Base64 format; it is passed when creating/renewing the Contact object. The first parameter is the name of the file + its extension; the second parameter - the image in Base64 format;
note - the additional information about the contact;
owner_guid - (optional) the globally unique identifier of the object User who created current contact. You are allowed not to pass this parameter, in this case the globally unique identifier of current User object will be used as the owner's one;
contact_lists - the globally unique identifiers of the contact lists to which the contact belongs;
phone - (optional) the telephone number of the contact stored as a dictionary in the remark-value format. Each of the values can include other remark-value pairs;
email - (optional) the E-mail address of the contact stored as a dictionary in the remark-value format. Each of the values can include other remark-value pairs;
address - (optional) the address of the contact stored as a dictionary in the remark-value format. Each of the values can include other remark-value pairs.

5.3.2 ProContact API methods

Method	Description
create_contact(Contact)	Create new Contact object; return the globally unique identifier of it. Contact - a Contact object in JSON format.
	create_contact(Contact) : ['success', contact_guid]
	For example:

	create_contact({
	"prefix_name" : "mr.",
	"first_name" : "Nick",
	"middle_name" : "",
	"last_name" : "Hammond",
	"suffix_name" : "",
	"birthday" : 12312321321,
	"company" : "VDOM",
	"position" : "developer",
	"picture_data" : {“name”: "Photo.jpg",“data”: "TWFuIGlzIGRpc3Rpbmd1aXNoZW" },
	"note" : "note to contact",
	"contact_lists" : ["8d74d349-17e4-401d-aa38-b8dcecd4e223", "5d74d349-15e4-401d-aa38-b8dcecd4e223"],
	"phone" : [{
	"remark" : "home",
	"value" : "8-909-545-4545" ,
	},
	{
	"remark" : "work",
	"value" : "8-509-545-4545"
	}
	] }
	"email" : [{
	"remark" : "home",
	"value" : "lena@home.ru" ,

Method	Description
	```
	    },
	    {
	        "remark" : "work",
	        "value" : "lena@work.ru"
	    }
	    ] }
	    "address" : [ {
	        "remark" : "home",
	        "value" : {"postal_index": "635412", "city": "Tomsk", "region": "tomsk obl", "country": "Russia",
	"address": "krasnoarmeyskaya"}]
	} )
	```

create_contactlist(ContactList)	Create new ContactList object; return the globally unique identifier of it.
	ContactList - a ContactList object in JSON format..
	
	create_contactlist(ContactList) : ['success', contactlist_guid]
	
	For example:
	```
	create_contactlist( {
	    "name"   : "Parents",
	    "color"    : "#F00"
	} )
	```

delete_contacts(contacts_guids)	Delete the Contact objects; return an empty String if the objects have been deleted successfully.
	contacts_guids - the globally unique identifiers of the Contact objects in JSON format.
	
	delete_contacts(data) : ['success']
	
	For example:
	```
	delete_contacts( {
	    "contacts_guids" :  [ "23", "45" ]
	} )
	```

delete_contactlists(contactlists_guids)	Delete the ContactList objects; return an empty String if the objects have been deleted successfully.
	contactlists_guids - the globally unique identifiers of the Contact objects in JSON format.
	
	delete_contactlists(data) : ['success']
	
	For example:
	```
	delete_contactlists( {
	    "contactlists_guids" :  [ "8d74d349-17e4-401d-aa38-b8dcecd4e223", "5d74d349-15e4-401d-aa38-
	b8dcecd4e223" ]
	} )
	```

login(data)	Login the application, return the User object. data - a User object in JSON format.

Method	Description

	`login( data ) : [ 'success', User ]`
	
	For example:
	
	`login( {`
	`    "login"    : "login",`
	`    "password" : "password"`
	`} )`

retrieve_contacts	Retrieve the Contact objects; return an Array consisting of them.
	contactlists_guids - the globally unique identifiers of the ContactList objects;
	contacts_guids - the globally unique identifiers of the Contact objects;
	query - a query string used for searching among the Contact objects. You can search by the following parameters: "prefix_name", "mr.", "first_name", "middle_name", "last_name", "suffix_name", "company", "position", "additional_info".
	
	`retrieve_contacts( data ) : [ 'success', [ Contact, Contact ] ]`
	
	For example:
	
	`retrieve_contacts( {`
	`    "query"  : "mike",`
	`    "contactlists_guids" : [ "8d74d349-17e4-401d-aa38-b8dcecd4e223" ]`
	`} )`

retrieve_contactlists	Retrieve the ContactList objects; return an Array consisting of them.
	contactlists_guids - the globally unique identifiers of instances of a ContactList class;
	access - the access rights: 'w' - (default value) write rights; 'r' - read rights; 'o' - owner of the object.
	
	`retrieve_contactlists( data ) : [ 'success', [ ContactList, ContactList ] ]`
	
	For example:
	
	`retrieve_contactlists( {`
	`    "access"  : "o"`
	`} )`

update_contact(Contact)	Update the Contact object (renew its picture or put the Contact object to the contact list); return a globally unique identifier of it. Contact - a Contact object in JSON format. `update_contact( Contact ) : [ 'success', contact_guid ]`
update_contactlist(ContactList)	Update the ContactList object (renew its name or its color); return a globally unique identifier of it. ContactList - a ContactList object in JSON format. `update_contactlist( ContactList ) : [ 'success', contactlist_guid ]`

5.3.3 Errors description

Error code	Error name	Description	Your action
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the objects' structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh does not exist.	Refresh the list of objects.
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.

5.4 ProPlanning API

5.4.1 ProPlanning objects

5.4.1.1 User

User is an object of the ProPlanning application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user.

5.4.1.2 Calendar

Calendar is an object of the ProPlanning application that stores information about the calendars of the application. It has the following parameters:
guid - the globally unique identifier of the calendar; it is not passed when creating new Calendar object;
name - the name of the calendar;
color - the color of the calendar;
access - (optional) the access rights; it is not passed when creating new Calendar object: 'w' - (default value) write rights, allows creating and editing events, creating and editing calendars, but does not allow deleting them; 'r' - read rights, allows only look through the events and calendars; 'o' - owner of the object, allows any manipulations with events and calendars;
events - (optional) the globally unique identifiers of the Event objects belonging to current calendar; it is not passed when creating/renewing the Calendar object.

4.4.1.3 Event

Event is a subobject of the Calendar object of the ProPlanning application that stores information about the events. It has the following parameters:
guid - the globally unique identifier of the event; it is not passed when creating new Event object;
start_date - the start date of event in the UnixTimeStamp format;

end_date - the end date of event in the UnixTimeStamp format;
summary - the name of the event;
description - (optional) the description of the event;
all_day - (optional, by default - True) the duration of the event is all-day;
rrule - (optional, by default - nothing) the recurrence rule of the event; it is set in iCalendar format;
creator_guid - (optional) the globally unique identifier of the object User who created current event; it is not passed when creating new Event object;
exdates - (optional, by default - empty) the exception dates when the event is not repeated; it is set together with the recurrence rule ("rrule" parameter);
calendar_guid - the globally unique identifier of the Calendar object, to which current event belongs;
editable - (optional) the access rights for editing the event. If it is set to True, then the access rights are inherited from an instance of the Calendar object, to which current even belongs. If it is set to False, editing and deleting the event is forbidden. Usually False indicates that the event represents a link to another event. This happens when user accepts an invitation to the event. This parameter is not passed when creating new Event object;
invites - the globally unique identifier of the Invite object; it is passed only when creating new Invite object. The "invites" parameter will not be passed by the "retrieve_events" method.

5.4.1.4 Invite

Invite is an object of the ProPlanning application that stores information about the invites. It has the following parameters:
guid - the globally unique identifier of the invite; it is not passed when creating new Invite object;
from_user_guid - (optional) the globally unique identifier of the object User who sends the invite; it is not passed when creating/renewing the Invite object;
to_user_guid - (optional) the globally unique identifier of the object User who receives the invite; if the "to_email" parameter is set, the "to_user_guid" parameter is not passed when creating new Invite object;
to_email - (optional) the E-mail address of the User object who receives the invite; if the "to_user_guid" parameter is set, the "to_email" parameter is not passed when creating new Invite object;
event_guid - (optional) the globally unique identifier of the Event object, to which the invite refers; it is not passed when creating/renewing the Invite object;
summary - (optional) the description of the invite; it is not passed when creating/renewing the Invite object;
start_date - (optional) the start date of an event to which current invite refers in the UnixTimeStamp format. This parameter copies the value of the "start_date" parameter with the globally unique identifier specified by the "event_guid" parameter;
status - (optional) the status of the invite. There is four possible variants of value: 0 - accepted; 1 - rejected; 2 - sent; 3 - accepted by E-mail (the invite is still active and can be accepted or rejected); it is not passed when creating new Invite object;
creation_date - (optional) the date of creating the invite; if the "to_user_guid" parameter is set, the "creation date" is not passed when creating new Invite object.

5.4.2 ProPlanning API methods

Method	Description
create_calendar	Create new Calendar object; return the globally unique identifier of it. Calendar - a Calendar object in JSON format. `create_calendar( Calendar ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
create_event	Create new Event object; return the globally unique identifier of it. If the "invites" parameter is not empty, the Invite objects will be created. Event - an Event object in JSON format. `create_event( Event ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
delete_calendars	Delete the Calendar objects; return an empty String if the objects have been deleted successfully. The

Method	Description
	Calendar objects to which you have access rights will be deleted. calendars_guids - globally unique identifiers of the Calendar. `delete_calendars( data ) : [ 'success' ]` For example: `delete_calendars( { "calendars_guids" : [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ] } )`
delete_events	Delete the Event objects; return an empty String if the objects have been deleted successfully. The Event objects to which you have access rights will be deleted. calendars_guids - globally unique identifiers of the Calendar objects. `delete_events( data ) : [ 'success' ]` For example: `delete_events( { "events_guids" : [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ] } )`
retrieve_calendars	Retrieve the Calendar objects; return an Array consisting of them. calendars_guids - globally unique identifiers of the Calendar objects; access - the access rights: 'w' - write rights; 'r' - (default value) read rights; 'o' - owner of the object. `retrieve_calendars ( data ) : [ 'success', [Calendar, Calendar] ]` For example: `retrieve_calendars ( { "calendars_guids" : [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ], "access" : "w" } )`
retrieve_events	Retrieve the Event objects; return an Array consisting of them. calendar_guids - globally unique identifiers of the Calendar objects. `retrieve_events( data ) : [ 'success', [ Event, Event ] ]` For example: `retrieve_events( { "calendar_guids" : [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ] } )`
retrieve_invites	Retrieve the Invite objects; return an Array consisting of them. When calling this method without

Method	Description

	arguments - all Invite objects will be returned.
	event_guid - globally unique identifiers of the Invite objects;
	`retrieve_invites( data ) : [ 'success', [ Invite, Invite ] ]`
	For example:
	`retrieve_invites( {`
	`    "event_guid" : "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF"`
	`} )`

update_calendar	Return the globally unique identifier of the Calendar object. Calendar - a Calendar object in JSON format. `update_calendar( Calendar ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
update_event	Return the globally unique identifier of the Event object. If the "invites" parameter is not empty, the Invite objects with an empty "guid" parameter will be created. Event - an Event object in JSON format. `update_event( Event ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`

update_invite	Update the Invite object (the "status" and the "calendar_guid" parameters); return an empty String.
	guid - a globally unique identifier of the Invite object;
	status - the status of the invite. There is four possible variants of value: 0 - accepted; 1 - rejected;
	calendar_guid - the globally unique identifier of the Calendar object, to which current Invite object belongs.
	`update_invite( data ) : [ 'success', '' ]`
	For example:
	`update_invite( {`
	`    "guid" :  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF",`
	`    "status" : "1",`
	`    "calendar_guid" : "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF"`
	`} )`

5.4.3 ProPlanning recurrence rule

The ProPlanning recurrence rule defince s the way of repetition of events in ProPlanning. The format of iCalendar© (http://www.ietf.org/rfc/rfc2445.txt) is used here.

There are some of the examples of setting the recurrence rule in ProPlanning:

Repeat mode	Notation	Description
Daily repeat №1	FREQ=DAILY;BYDAY=MO,TU,WE,TH,FR	Every workday.
Daily repeat №2	FREQ=DAILY;INTERVAL=2	Every second day.
Weekly repeat №1	FREQ=WEEKLY;INTERVAL=2	Every second week.
Weekly repeat №2	FREQ=WEEKLY;BYDAY=MO,TU	Every week on Mondays and Tuesdays.
Monthly repeat	FREQ=MONTHLY;INTERVAL=2;BYMONTHDAY=3	Every 3rd day of every second month.

Repeat mode	Notation	Description
№1		
Monthly repeat №2	FREQ=MONTHLY;NTERVAL=2;BYDAY=3MO	Every 3rd Monday of every second month.
Annual repeat №1	FREQ=YEARLY;INTERVAL=1;BYMONTHDAY=2;BYMONTH=2	Every 2nd February of every year.
Annual repeat №2	FREQ=YEARLY;INTERVAL=1;BYDAY=2TU;BYMONTH=2	Every 2nd Tuesday of February of every year.

5.4.4 Errors description

Error code	Error name	Description	Your actions
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the objects' structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh does not exist.	Refresh the list of objects.
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.
8	errCalendarObjNotExist	The Calendar object you want to pass / refresh does not exist.	Refresh the list of objects.
9	errEventObjNotExist	The Event object you want to pass / refresh does not exist.	Refresh the list of objects.
10	errInviteObjNotExist	The Invite object you want to pass / refresh does not exist.	Refresh the list of objects.

5.5 ProSearch API

5.5.1 ProSearch objects

5.5.1.1 User

User is an object of the ProSearch application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user.

5.5.1.2 Agent

Agent is an object of the ProSearch application that stores information about the agent of the application. Agent is a separate application that sends information to the ProSearch application. It has the following parameters, which are passed on creating the Agent object.

guid - the globally unique identifier of the agent;
name - the name of the agent.

5.5.1.3 Index

Index is an object of the ProSearch application that stores information about the index of the application. Index represents information that is sent by agent to ProSearch. It has the following parameters, which are passed on creating the Index object.
id - the identifier of an index;
name - the name of the file;
search_source - the path where the information about the file will be stored;
data - the text information about the file;
url - the URL address by which the file will be accessed;
agent_guid - the globally unique identifier of the Agent object.

5.5.1.4 SearchSource

SearchSource is an object of the ProSearch application that stores information about the search source of the application. Search source represents a place where the ProSearch application searches information. It can be a path to a folder if you search some documents or another ProSuite application if you search contacts in ProContact or files in ProShare. It has the following parameters:
search_source - the place where to search;
agent_guid - the globally unique identifier of the Agent object.

5.5.1.5 Document

Document is an object of the ProSearch application that stores information about the documents stored by of the application. Here document represents not only text files, but also the contacts in ProContact and files in ProShare. It has the following parameters:
guid - the globally unique identifier of the document;
filename - the name of the document;
content - the content of the document;
path - the search source where the document is stored;
physical_name;
creation_datetime;
author.

5.5.2 ProSearch API methods

Method	Description

create_agent	Create new Agent object; return the globally unique identifier of it.
	Agent - an Agent object in JSON format.
	`create_agent( Agent ) : [ 'success', "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	For example:
	`create_agent( {`
	`    "guid"     : "com.new_agent",`
	`    "name"   : "ProSearch Agent"`
	`} )`

create_index	Create new Index object; return the globally unique identifier of it. Index - an Index object in JSON format.

Method	Description
	create_index(Index) : ['success'] For example: create_index({ "id" : "123", "name" : "User Userovich", "search_source" : "search_source_path", "data" : "LastName=Userovich;FirstName=User", "url" : "http://ProContact.com/home?guid=1405", "agent_guid" : "com.new_agent" })
create_search_source	Create new SearchSource object; return the globally unique identifier of it. SearchSource - a SearchSource object in JSON format. create_search_source(SearchSource) : ['success'] For example: create_search_source({ "search_source" : "search_source_path", "agent_guid" : "com.new_agent" })
delete_agent	Delete the Agent objects; return an empty String if the objects have been deleted successfully. agent_guid - globally unique identifier of the Agent object. delete_agent(data) : ['success'] For example: delete_agent({ "agent_guid" : "com.new_agent" })
delete_index	Delete the Index objects; return an empty String if the objects have been deleted successfully. Index - an Index object in JSON format. delete_index(Index) : ['success'] For example: delete_index({ "id" : 123, "name" : "index name", "search_source" : "search_source_path", "agent_guid" : "com.new_agent" })
delete_search_source	Delete the SearchSource objects; return an empty String if the objects have been deleted

Method	Description

	successfully.
	
	SearchSource - a SearchSource object in JSON format.
	
	`delete_search_source( SearchSource ) : [ 'success' ]`
	
	For example:
	
	`delete_search_source( {` `"search_source"   : "search_source_path",` `"agent_guid"     :   "com.new_agent"` `} )`

login(data)	Login the application, return the User object.
	data - a User object in JSON format.
	
	`login( data ) : [ 'success', User ]`
	
	For example:
	
	`login( {` `"login"    : "login",` `"password" : "password"` `} )`

retrieve_results	Retrieve the Document objects; return an Array consisting of them.
	query - the text string for searching;
	access - the access rights: 'w' - write rights; 'r' - (default value) read rights. You can also set the directories where to search.
	
	`retrieve_results( data ) : [ 'success', [ Document, Document ] ]`
	
	For example:
	
	`retrieve_results( {` `"query"    : "user"` `} )`

5.5.3 Errors description

Error code	Error name	Description	Your actions
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the objects’ structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh	Refresh the list of objects.

Error code	Error name	Description	Your actions
		does not exist.	
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.
8	errObjectAlreadyExist	The object you want to create already exists.	
9	errAgentLimitation	You have reached the limit of Agent objects.	
10	errSourceLimitation	You have reached the limit of SearchSource objects.	
11	errAgentObjectAlreadyExist	The Agent object with such globally unique identifier already exists.	
12	errSearchSourceObjectAlreadyExist	The SearchSource object with such name already exists.	
13	errAgentObjNotExists	The Agent object with such globally unique identifier does not exist.	
14	errSearchSourceObjNotExists	The SearchSource object with such name does not exist.	

5.6 ProMail API

5.6.1 ProMail objects

5.6.1.1 User

User is an object of the ProMail application that stores information about the users of the application. It has the following parameters:
guid - the globally unique identifier of the user; it is not passed when creating new User object;
email - the E-mail of the user; it is used as login;
first_name - the first name of the user;
last_name - the last name of the user.

5.6.1.2 Archive

Archive is an object of the ProMail application that stores information about the archives of the application. It has the following parameters:
guid - (optional) the globally unique identifier of the archive, it is not passed when creating the Archive object;
name - the name of the archive;
color - (optional) the color of the archive, by default the value is #FFFFFF;
mailbox_guid - the globally unique identifier of the Mailbox object, where the archive will be created.

5.6.1.3 Mailbox

Mailbox is an object of the ProMail application that stores information about the mailboxes. It has the following parameters:
guid - (optional) the globally unique identifier of the mailbox.
name - the name of the mailbox;
address - the E-mail address of the mailbox.

5.6.1.4 Mail

Mail is an object of the ProMail application that stores information about the mails. It has the following parameters:
guid - (optional) the globally unique identifier of the mail, it is not passed when creating the Mail object;
subject - the E-mail subject;
from_email - the E-mail address from which the letter is sent;
to_email - the E-mail address to which the letter is sent;
date - (optional) the date and time when the letter is sent. If this parameter is not indicated explicitly, the time of the server will be used.
priority - the priority of the letter (by default - normal);
location - the location where to put the letter;
body - the main part (body) of the letter;
attachment - the attachments to the letter, the number of them is in output;
archive -

5.6.1.5 Attachment

Attachment is an object of the ProMail application that stores information about the attachments to letters. It has the following parameters:
guid - (optional) the globally unique identifier of the attachment, it is not passed when creating the Attachment object;
filename - the name of the attachment;
download_url - (optional) the link to the attachment, it is not passed when creating the Attachment object;
mimetype - the MIME type of the attachment;
filedata - the attachment encoded in base64 format.

5.6.2 ProMail API methods

Method	Description

create_archive	Create new Archive object for the current user; return the globally unique identifier of it.
	Archive - an Archive object in JSON format.
	
	`create_archive( Archive ) : [ 'success', archive_guid ]`
	
	For example:
	
	`create_archive( {`
	`    "name"   : "VDOM Box Research",`
	`    "color"    : "#FFFFFF",`
	`    "mailbox_guid" : "123"`
	`} )`

create_mailbox	Create new Mailbox object for the current user; return the globally unique identifier of it.
	Mailbox - a Mailbox object in JSON format.
	
	`create_mailbox( Mailbox ) : [ 'success', mailbox_guid ]`
	
	For example:
	
	`create_mailbox( {`
	`    “name”      : “BgTronic.Directeur_box”,`
	`    “email”     : “BgTronic.Directeur@cybertronique.com”,`
	`    "password"  : "password"`
	`} )`

Method	Description

delete_archive	Delete the Archive objects; return an empty String if the objects have been deleted successfully. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	guid - globally unique identifier of the Archive object.
	`delete_archive( data ) : [ 'success' ]`
	For example:
	`delete_archive( {`
	`    "guid" :  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF"`
	`} )`

delete_mail	Delete the Mail objects from the mailbox; return an empty String if the objects have been deleted successfully. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	mailbox_guid - globally unique identifier of the Mailbox objects;
	mails_guids - globally unique identifiers of the Mail objects.
	`delete_mail( data ) : [ 'success' ]`
	For example:
	`delete_mail( {`
	`   "mailbox_guid" :  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF",`
	`    "mails_guids"  :  [ "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	`} )`

delete_mailbox	Delete the Mailbox objects; return an empty String if the objects have been deleted successfully. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	mailbox_guid - globally unique identifier of the Mailbox. object
	`delete_mailbox( data ) : [ 'success' ]`
	For example:
	`delete_mailbox( {`
	`    "mailbox_guid" :  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF"`
	`} )`

fetch_mails_from_pop3	Fetch the data from POP3 sever; return nothing. `fetch_mails_from_pop3( )`
login(data)	Login the application, return the User object. data - a User object in JSON format. `login( data ) : [ 'success', User ]`

Method	Description

	For example:
	`login( {`
	`    "login"    : "login",`
	`    "password" : "password"`
	`} )`

retrieve_archives	Retrieve all Archive objects by the globally unique identifiers of the Mailbox objects; return an Array consisting of them. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	mailboxes_guids - the globally unique identifiers of the Mailbox objects.
	`retrieve_archives( data ) : [ 'success', [ Archive, Archive ] ]`
	For example:
	`retrieve_archives( {`
	`    "mailboxes_guids" : [  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	`} )`

retrieve_mailboxes	Retrieve all Mailbox objects by their globally unique identifiers; return an Array consisting of them. The method can be called without parameters, in this case all data of new the Mailbox objects will be returned. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	mailboxes_guids - globally unique identifiers of the Mailbox objects.
	`retrieve_mailboxes( data ) : [ 'success', [ Mailbox, Mailbox] ]`
	For example:
	`retrieve_mailboxes( {`
	`    "mailboxes_guids" : [  "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF" ]`
	`} )`

retrieve_mails	Retrieve all Mail objects by the globally unique identifier of the Mailbox object; return an Array consisting of them. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur. mailbox_guid - globally unique identifier of the Mailbox object; mail_guid - globally unique identifier of the Mail object; archive_guids - globally unique identifier of the Archive object; limit - the number of objects to return; offset - the number indicating the displacement from the beginning and up to the given number; query - a query string used for searching among the Mail objects. `retrieve_mails( data ) : [ 'success', [ Mail, Mail ] ]` For example:

Method	Description

	retrieve_mails({
	    "mailbox_guid" : "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF",
	    "limit" : 10,
	    "offset" : 0,
	    "query" : "Work"
	})

retrieve_new_mails_count(data)	Retrieve the Mailbox objects; return the output in JSON format. The method can be called without parameters, in this case all data of the Mailbox objects will be returned. mailboxes_guids - globally unique identifiers of the Mailbox objects. retrieve_new_mails_count(data) : ['success', [output, output]] For example: retrieve_new_mails_count({ "mailboxes_guids " : ["6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF"] })
retrieve_senders	Retrieve all E-mail addresses of senders; return an Array consisting of them. No parameters are needed. retrieve_senders( ) : ['success', ["email@email.com", "user@user.com"]]
send_mail(data)	Send E-mail; return the globally unique identifier of the Mail object. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur. mailbox_guid - a globally unique identifier of the Mailbox object; mail - a Mail object. send_mail(data) : ['success', mail_guid] For example: send_mail({ "mailbox_guid" : "6F6F6F6F-8B8B-D1D1-B4B4-CFCFCFCFCFCF", "mail" : Mail })
update_archive	Update the Archive object; return the globally unique identifier of it. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur. Mailbox - an Archive object. update_archive(Archive) : ['success', archive_guid] For example: update_archive({ "guid" : "10", "name" : "VDOM Box Research",

Method	Description
	"color" : "#FFFFFF", "mailbox_guid" : "123" })

update_mailbox	Update the Mailbox object; return the globally unique identifier of it. When executing this method an errNotNeededRules error, that user does not have access rights to perform such operation, may occur.
	Mailbox - a Mailbox object.
	`update_archive( Mailbox ) : [ 'success', mailbox_guid ]`
	For example:
	update_archive({
	"guid" : "10",
	"name" : "BgTronic.Directeur_box",
	"email" : "BgTronic.Directeur@cybertronique.com",
	"password" : "password"
	})

5.6.3 Errors description

Error code	Error name	Description	Your action
0	errScriptError	An error has occurred during the script execution.	Report to a developer.
1	errBadJSONFormat	The JSON object has a syntax error and cannot be processed correctly.	Check the compliance with the JSON objects' structure.
2	errNotLoggedIn	The session has expired and you have been logged off.	Re-login the system.
3	errBadObjectFormat	The JSON object has wrong structure.	Check the object structure.
4	errObjNotExists	The object you want to pass / refresh does not exist.	Refresh the list of objects.
5	errNotNeededRules	You do not have the necessary rights to edit the object.	Check the list of user's rights for this object or refresh the list of objects.
6	errEmptyPassword	When trying to log in an empty password has been passed.	Enter the password and re-login the system.
7	errLoginError	Wrong login and password.	Check the login and password.
8	errArchiveObjectSameNameExist	The Archive object with such name already exists.	
9	errMailboxObjectNotExist	The Mailbox object with such globally unique identifier does not exist.	
10	errArchiveObjectNotExist	The Archive object with such globally unique identifier does not exist.	

6 Additional information

The additional information on creating macro’s source code in VScript programming language you can find in the VScript Documentation.

Index