Important Dialogs

The following is a description of the main application dialog boxes.

Breakpoint Manager

This dialog box manages the active project breakpoints status. A breakpoint is a mark added to aid debugging. When the application reaches that line, it will interrupt the execution of the program. The line containing the breakpoint will not be executed.

To add or remove a breakpoint in the active document, move the caret to the line and press the "Inserts/Remove Breakpoint" button. The breakpoint is displayed in the editor as a maroon circle on the left of the line. When a breakpoint is added, it is active for all the threads of the program. By default there is no condition for the break, i.e. the execution will break in all cases. It is possible to add a break condition, using the "Condition to stop execution", field, which is explained below.

The *Breakpoint Manager* dialog box shows a table with information on all the project breakpoints. There is a row for each breakpoint. The following table explains each component of the *Breakpoint Manager* dialog box:

Table 4.17. Breakpoint Manager Dialog Box

Menu Item	Description
'*Enabled*' *column*	Shows if the breakpoint is active or not. It can be toggled.
'*File path*' *column*	Full file path of the file that contains the breakpoint.
'*File line*' *column*	The line in the source code file were the breakpoint is.
'*Thread ID*' *column*	Thread that will "see" the breakpoint. Valid data includes: 'all', '1', '2', '3', '4', etc.
'*Condition to stop execution*' *column*	Condition that will be evaluated when the execution get the breakpoint. If the condition evaluates to be true, the execution will be interrupted. If the condition evaluate to be false, the execution will continue uninterrupted. This is a powerful feature and is fully explained in the section called “Expression evaluation help”
'*Show source code for selected breakpoint*' *button*	When this button is pressed, the editor opens the file and shows the line where the selected breakpoint is located.

Expression evaluation help

Expression data types

- boolean
- integer
- string

Expression operators

The following is a list of the expression operators available in SIMPOL

Table 4.18. Expression operators

Operator	Syntax	Return Type
+	integer + integer	integer
-	integer - integer	integer
*	integer * integer	integer
/	integer / integer	integer
<	integer < integer	boolean
>	integer > integer	boolean
<=	integer <= integer	boolean
>=	integer >= integer	boolean
==	integer == integer	boolean
!=	integer != integer	boolean
<>	integer <> integer	boolean
+	string + string	string
==	string == string	boolean
!=	string != string	boolean
<>	string <> string	boolean
and	boolean and boolean	boolean
or	boolean or boolean	boolean
()	Used to group sub-operations	none

NB: != and <> are equivalent

Constant Values

Data Type	Accepted Values
boolean	`.true` or `.false`
integer	Any integer `>= -2147483647` and `<= 2147483647`
string	Any array of characters in between quotes: `"....."` or `'.....'`

Variable Values

- A variable value is a function local variable or a type property value

Breakpoint Condition

- Must be a boolean expression

Watch Window Expression

- Can be any expression or object reference

Built In Functions

The following is a list of the built-in functions in the SIMPOL language:

Syntax	Return Type	Example	Returned Value
`#StrAscii (string input)`	string	`#StrAscii ("Hello")`	`68 65 6C 6C 6F`
`#StrUnicode (string input)`	string	`#StrUnicode ("Hello")`	`0068 0065 006C 006C 006F`
`#StrLength (string input)`	integer	`#StrLength ("He{d}{a}llo")`	`11`
`#StrLengthEx (string input)`	integer	`#StrLengthEx ("He{d}{a}llo")`	`7`
`#Trace (string text)`	`.false`	`Example: #Trace ("Hello")`	`.false (the text goes to the debug window)`
`#BoolToStr (boolean input)`	string	`#BoolToStr (.true)`	`".true"`
`#IntToStr (integer input)`	string	`#IntToStr (123)`	`"123"`
`#StrUCase (string input)`	string	`#StrUCase ("Hello")`	`"HELLO"`
`#StrLCase (string input)`	string	`#StrLCase ("Hello")`	`"hello"`
`#StrTrim (string input)`	string	`#StrLCase (" Hello ")`	`"Hello"`

Syntax	Return Type	Example	Returned Value
`#StrRight (string input, integer length)`	string	`#StrRight ("Hello", 2)`	`"lo"`
`#StrLeft (string input, integer length)`	string	`#StrLeft ("Hello", 2)`	`"He"`
`#StrMid (string input, integer from, integer length)`	string	`#StrMid ("Hello", 3, 2)`	`"lo"`
`-`	`-`	`#StrMid ("Hello", 3, -2)`	`"el"`
`#StrFindF (string input, integer from, string match)`	integer	`#StrFindF ("Hello", 0, "el")`	`1`
`-`	`-`	`#StrFindF ("Hello", 0, "bye")`	`-1 (not found)`
`#StrFindB (string input, integer from, string match)`	integer	`#StrFindB ("Hello", 5, "el")`	`1`
`-`	`-`	`#StrFindB ("Hello", 5, "bye")`	`-1 (not found)`
`#FileTrace (string path, string text)`	`.false`	`#FileTrace ("c:\temp\log.txt", "Hello")`	`.false (The text is appended to the file "c:\temp\log.txt")`
`#StrHextToInt (string input)`	integer	`#StrHextToInt ("FFFE")`	`65534`
`#IntToStrHext (integer input)`	string	`#IntToStrHext (65534)`	`"FFFE"`
`#StrRepeat (string input, integer repeat)`	string	`#StrRepeat ("Hello", 3)`	`"HelloHelloHello"`
`#VarContentToFile (string path, string variable)`	`.false`	`#VarContentToFile ("c:\temp\log.txt", "MyObject.property1")`	`.false (Useful for analyzing the content of large variables)`
`#VarContentLength (string variable)`	integer	`#VarContentLength ("MyObject.property1")`	`12000`
`#VarStructToFile (string path, string variable)`	`.false`	`#VarStructToFile ("c:\temp\log.txt", "MyObject")`	`.false (The object structure and content is saved in XML format)`
`#GetCurrentDirectory()`	string	`#GetCurrentDirectory()`	`"c:\temp\"`

Breakpoint Expression Examples

Below are some sample code fragments and some example breakpoint expressions. Breakpoint expressions can only be evaluated on lines of executable code.

```
type TA
```

```
  bool m1
  integer m2
  string m3
end type

type TB
  string m4
  TA m5
end type

function example()
  bool b
  integer i
  string s
  TB tb

  ............
  ............
end function
```

Table 4.19. Breakpoint Expression Examples

Sample	Return Type
`1 < i and s == "Hi"`	boolean
`(1 + 3) * 2 + i >= 3`	boolean
`b and .false or tb.m5.m1`	boolean
`tb.m5.m1`	boolean
`s`	string
`tb`	`object reference`
`(b or tb.m5.m1) and tb.m4 == s`	boolean
`(i * 3 + 2) + 2 - 14 / tb.m5.m2`	integer

Call Analyzer

This is only available if there is an active project. This dialog box represents a tree view of all the calls among functions in the project. Each node represents a function. If the dialog box is opened from "Menu/Tools/CallGraph", the parent function calls the child one. If the dialog box is opened frome "Menu/Tools/CallerGraph", the parent function is called by the child one.

The dialog box has a combo box for the user to select the project function that will be the root of the tree. Once this is selected, the tree will be created with the functions that call or get called by the parent node and so on.

Each node displays the name of the function that it represents. At the bottom of the dialog box the file path of the function is displayed and the function prototype. If the user double clicks on a node, the editor will open the source code file where the function is located and show the function definition. This will happen only if the source code file is available.

Check Project File

This is only allowed if there is no active project. Using this dialog box, the user can check if the structure of a project definition file (a `smj` file) is valid or not.

The dialog box has an entry for the project definition file path. After entering the file path, the button "Check consistency" becomes active. This button will check the consistency of the file and will display the result in the box below the button. There are two buttons below the result box. The first button edits the project file definition. The second button loads the project that will be active if the project definition file is OK. If there are problems with the project definition file, the user has to edit it, make the appropiate changes and save the modified file. After that the user must check the consistency again repeat the process until the project description file is valid.

This dialog box will always show up if the user tries to load a project with an invalid project desciption file.

Application Options

This dialog box stores general application properties. This information is stored in the windows registry.The following properties can be modified:

Property	**Description**
SIMPOL compiler file path	Shows the SIMPOL compiler file path. It is a `smp` file with the functionality to compile a source code file (`sma` or `smu`) into a byte-code file (`smp` or `sml`). The SIMPOL compiler is a byte-code file that is executed in the SIMPOL virtual machine as any other `smp` or `sml` file. If the application cannot find the compiler file at the path you specify, it will search in the root directory of the application.
Working Directory	Use this text box to enter the path of the folder you wish to contain your Superbase NG projects. This information will be used by the application to make it easier for the user to handle project information.

Property	Description
MultiLanguage	If this checkbox is checked, the user will be able to work with several different languages in the application. This means the user will be able to edit different types of files, and different color-codig syntax rules will be applied in the IDE. If MultiLanguage is not checked, the application will work only with the SIMPOL language.
Optimize Linker Output	This checkbox affects the project link process. If it is not checked, the application will concatinate all the `smp` and `sml` files. This set of files includes the result of compiling all the project modules, plus all the files that are in the list of files to link. Any of the files to link is the result of an external project build, so it is quite common that the files to link share part of its content among them. If we check this option, the application will analyze all the files to concatenate and it will remove all the repeated information in the final project byte-code file result of the project build.
Create application Icon File associations	If this button is pressed, a process will be launched to create the application icon file associations. This process makes appropriate changes to the registery, that allow the operating system to associate an icon for each of the following file extensions: `sma`, `smu`, `smz` and `smj`. Hence, if the user double-clicks on any of these files (in windows explorer etc.) the file will be opened with the SIMPOLIDE application. Options to build, rebuild or execute are displayed in a popup menu if the user right-clicks on the icon of a `smj` file.
Languages	In this area, the user can add, remove or change the properties of all the languages the application can handle. The description of each language is stored in a diferent ."`ini`" file. By default the application handles the following languages: Binary, C++, C#, HTML, IDL, JScript, SIMPOL, Superbase, Text, Visual Basic, VBScript and XML. A language can be activated or deactivated with the "active" checkbox. On the left of this area there is a list of languages. On the right, the path to the selected language description file is displayed and the file extensions associated to the language. If there is more than one, they have to be separated by semicolons. There is also a short description of the selected language.To edit the selected language settings, click on the "Edit Language..." button.

Property	Description
Help	There is a checkbox to activate the help system. The help system is called by pressing "F1" button from any place in the application. There are two entries in the help area. In the first entry, the user enters the program that will be opened when the help is invoked. The second entry is the file to open with that program. Example: "Exe File: `C:\Program Files\Internet Explorer\iexplore.exe`", "Command Line: `C:\SIMPOL\docs\index.html`".
Autosave	If this is selected, the application will save anu active project files and folders to the windows temporary folder.In this area, there are two entries. The first is to indicate how many minutes to wait between autosaves. The second entry is to indicate the maximun number of different copies to be stored in the temporay folder. For example, if the number of minutes is set to 15 and the maximun number of copies is set to 4, after working with "MyProject" for two hours, there will be 4 copies of the project in the windows temporary folder. The names will be: `MyProject_AutoSave_1`,`MyProject_AutoSave_2`,`MyProject_AutoSave_3` and`MyProject_AutoSave_4`, with the most recent being `MyProject_AutoSave_1`.
Save project documents before build...	If this box is checked and the user is working in a project, the documents that belong to the project will be saved just before a project build, rebuild, execute or debug start, .
SMA source code file default preference	If this box is checked, the default source code file for new projects becomes the `sma` type. Any new projects opened will now, by default, use only `sma` files. If this box is unchecked, new projects will, by default, only use `smu` files. These settings can be changed once the project is created by using the *Project Settings* dialog box (Project/Project Settings...). This property will not take effect if the user has the ASCII-only application build.

Languages

This dialog box allows the user to change the settings for a language. The language settings are stored in a .`ini` file. For example, the language settings file of the XML language is `XML.ini`.

This dialog box has the following tabs:

Editor

In this area the user can change the basic aspects of a specific language.

We can personalize the following properties:

Property	Description
Tab size	This changes how far the caret jumps when the user presses tab.
Auto indent	If the user hits the return button with auto indent turned on, the caret will be automatically indented to the same position as the text begins on the line above. Until the user types some text, the indentation is temporary, so if the user opens another active window, the caret will be left aligned when he or she returns.
Show whitespace	If this is checked, the tabulators and whitespaces in the document will be displayed with special characters.(a floating decimal place for whitespace and a ">>" character for tabulators).
Virtual whitespace	If this box is unchecked, the user cannot position the caret after the last blank space or character that has been typed in a line. If it is checked, the user can position the caret anywhere in the editing window and begin typing. The application will fill empty spaces in the line with tablators as necessary.
Replace tabs	If this is checked, all tabulators in the document are replaced by equivelant whitespaces. This will have no visual effect on the document unless you have the "Show whitespace" box checked.
Match case	If it is checked, the editor parser will work in a case sensitive mode, if not, the editor parser works in a non-case sensitive mode.
Font face name	Displays the name of the font that is used in the document.
Font size	Displays the size of the font. The "Font Settings" button can be used to change these options.

Parser

The parser is in charge of recognizing the keywords, string patterns, operators, etc throughout the document text. This information is used by the editor to color the text, following the rules of the specific language. These settings are very important, as they effect how the color-coding of the language functions, so the user has to be completely sure before making a change, especially with the SIMPOL language.

The following properties can be edited:

Property	Description

Operators	Characters that are operators in the language. For example: `+ - *`
Delimiters	Characters that are delimiters in the language.
KWStartChars	Special characters that can be the first character of a keyword.
KWMiddleChars	Special characters that can be in the middle of a keyword.
KWEndChars	Special characters that can be at the end of a keyword.

Keywords

Keywords are special reserved words in a language. For example, in SIMPOL language, kewords include: `for`, `if`, `function` and `while`.

On the left there is a list with all the language keywords. Above the list there are buttons to add and remove keywords. On the right side there is a combobox. The combobox list contains the names of the all the different color groups that can be selected. The user can then choose a color group per keyword.

Colors

Each language has a different set of color groups. SIMPOL, for example, has the following groups: `Comment`, `Keyword`, `Number`, `Operator`, `String`, `SystemFunction`, `SystemType`, `Text`, `TextSelection` and `UserType`.

Each color group has a foreground color and a background color. These two colors can be changed using the appropiate buttons on the right side of the color area.

Property	**Description**
Operators	Characters that are operators in the language. For example: `+ - *`

New Project Options

This dialog box is used to create a new Superbase NG project. It is used to set the properties of the new project.

These are the options:

Property	Description
Project output type	This is the type of file that the build of a project will generate. It can be `smp` or `sml`. Both are byte-code files that will be run in the SIMPOL virtual machine.
Project source code type	This is the type of source code file that will be used in the project. It can be `sma`, which is an ASCII file, or `smu`, which is a Unicode file.
Project location	The folder where the new project will be created.
Project name	The name of the new project.
Wrapper over SIMPOL code file	This means that the project will be created wrapping the SIMPOL source code file selected. A project with one module will be created, and the SIMPOL source code file will be the module's main source code file.
Get properties from project	If this box is checked, the user has the opportunity to select a project. The new project will inherit the project properties of selected project.

Debug Execution Profile

This option is only available if there is an active project and the user is not debugging it. The information recorded from a debug session is displayed here. It is a very powerful feature that shows the developer the bottle-necks of his SIMPOL program and, as a result, he can remove them and improve the program's performance. At the top left of the dialog box there is a checkbox to enable or disable this feature. If it is enabled, execution in debug mode will go a bit slower in order to record the function calls, time spent in each function, and so on.

There is a box at the top where the user can enter a number of microseconds. This is the maximum amount of time that will be recorded for a statement being executed. The reason behind introducing this cutoff is that a multitasking operating system can pause the execution of a process in the middle of an statement. In this case the time the statement takes to be executed is actually its own time plus the time the microprocessor doing other things. So if we know, for example, that a part of our statement is going to last less than 500 microseconds, we can set this time as a cutoff. This will probably remove all the time the microprocessor is out of our process in the profile report, or at least, the majority of it.

The most important thing in the dialog box is the table, where the recorded information will be displayed after a debug sesion. There is a row per function. The columns of the table are explained below:

Column	Description
Library	The library file name where the function is.

Column	Description
Function	The name of the function. It can be a global function or an object method.
Call count	The number of times the function is called.
Full time (milliseconds)	The time spent in the function for all the calls. It takes into account the time spent executing the function statement, plus any time spent in functions that were called from within it.
Truncate time (milliseconds)	The "Full time" minus the time spent for each statement bigger that the cutoff.
Error time (milliseconds)	This is an indication of the +/- error in the timing of the function. The actual time taken will be somewhere between "Full time" minus "Error time" and "Full time" plus "Error time".
Block time (milliseconds)	The amount of time for which the execution was blocked. This happens, for example, when the SIMPOL program uses sockets, tables in a data base, etc.

Project Settings

This is only available if there is an active project. It displays the project properties, and allows the user to edit them. The project settings are stored always in a file with `smj` extension. The name of this file is the name of the project.

The dialog box shows the following information:

File Folders

This is a list of the folders that the SIMPOL compiler will use to find the included files. There are two types of included path: absolute file paths and relative file paths.

For example, in a lamda source code file, there could be a line like this:

```
include
"c:\projects\myproject\includes\MyFile1.sma"
```

Or a line like this:

```
include "MyFile1.sma"
```

The advantage in the second example is that the path in the source code is not made explicit. Note that the path can have the slash or back slash character depending on the operating system.

If the path is absolute, the compiler will use it and nothing more. If the path is relative, for example `MyFile1.sma`, the compiler will firstly use the folder where the file is being compiled to search for the included file. If the included file is not there, it will search for it in each of the "File folders" folders until it is found.

*`sml` Files to link

A list with the `sml` files that will be included in the output project file.

Targets

A target is a copy of the output project file plus a shebang line that is added at the beginning of the file. A target is typically the output of a CGI Project, and the shebang line is the path to the SIMPOL virtual machine program that will execute the byte-code file. The target file is typically called from the web server, and the web server will take the information from the shebang line to execute the file. The targets are created in the project build process.

Targets is a table with a row per target. There are buttons to add, edit and remove a target. The add and edit target buttons will open the target manager dialog box, where the user can add or modify a target. The *Targets* table has the following colums:

Column	Description
Activate	A checkbox to activate or deactivate the target creation in a project build.
Target	The target file path. E.g. `c:\Apache\bin\MyProject.smp`.
Shebang Line	The shebang line. E.g. `#!c:\SIMPOL\smpcgi32.exe{d}{a}`.

Command line	The parameters that will be passed to the "main" function when the project is executed. The parameters are separated by one or more whitespaces. If a parameter contains whitespaces, they should be entered within double quotes or between single quotes.
	Example:
	`function main prototype: main(string s, string`
	`command line: "hi  bye" 123`

Output file (`smp`, *`sml`)*	The byte-code file that is generated as a result of a project build. If the project has a "main" function, this output file will have a `smp` extension and could be executed alone in the SIMPOL virtual machine. If the project does not have a "main" function, the output file will have `sml` extension. In this case, the byte-code file acts as a library to be linked to by other projects at design-time, or as a library to be loaded dynamically by a running SIMPOL program.

Column	Description
Source code file preference	The default file extension that the application will use for this project when the user creates a new file or opens a file etc.
Make file	If this box is checked, the application will create two make files in the project build process. One is to be used over Windows platforms' (NMAKE facility) and the other is to be used over the Linux platform's (MAKE facility). This make file has the information to make a project build. The file time dependencies are taken into account when making a project build.

Column	Description

CGI Project	This contains a checkbox and an area to enter information. It is provided in the case the user wishes work with a CGI Project. If the box is checked, the project changes from a normal project to a CGI project. A CGI SIMPOL program is pretended to be called from a web server. These programs have a main function but with an special parameter. This parameter is a reference to a CGICall object that will transport all the information that the web server received from a browser call. The CGI SIMPOL program will perform an action depending on the brower request and will output HTML code embedded in SIMPOL strings to the CGICall object. This HTML code will be returned to the browser that made the call through the web server and will be displayed in the customer's browser as a HTML web page. So in the end, what the CGI programer needs is a way to build HTML web pages quickly and a way to modify the web page dynamically, depending on the specific browser request.
	Instead of working directly with the SIMPOL code and trying to imagine how the HTML code embedded in SIMPOL strings will turn out, the programmer can create SIMPOL server pages and work in them. A SIMPOL server page is a HTML page with blocks of SIMPOL code embed. The advantage of working with SIMPOL server pages is that they can be displayed in the HTML viewer of the SIMPOL application. The build process will create the SIMPOL source code associated, and it will be compiled as any other SIMPOL source code file.
	If the CGI Project checkbox is checked, a new folder entitled "Server Pages" is created as a child of every module folder in the Project View Tree. All the SIMPOL server pages that belong to the specific module will be displayed in the folder. A SIMPOL server page is a file with `smz` extension.
	In the CGI Project area the user can set the extension of the source code file that will be generated when compiling a server page. It can be `sma` or `smu`. There is an entry to set the output call format - the "CGICall.output" method that will output an string to the standard output. By default the format is: `cgi.output(%s + "{0D}{0A}"`,
	An example is shown below. It is assumed here that the CGICall object is named "cgi". The "%s" are placeholders for the HTML embedded as a string.
	Output CGI format:
	`cgi.output(%s + "{0D}{0A}", 1)`
	Line in a SIMPOL server page file:
	`<TH>Hello </TH>`
	Line in the source code file associated
	`cgi.output("<TH>Hello </TH>" + "{0D}{0A}", 1`

Target Manager

This dialog box is opened when the "add/edit target" button in the *Project Manager* dialog box is pressed.

On the left there is a list of target folders and on the right there is a list of shebang lines. Below each list is an edit box where the user can modify the information (target folder or shebang line). There are buttons to add the content of the edit box to the appropiate list, remove an entry in the list or add the content of the entry in the list to the edit box. The list entries are stored in the windows registry. There is also a checkbox to activate or deactivate the target. Finally, there is an entry in the target area to enter the target file name. Note: This is usually the name of the project output, but it can be overwritten.

Watch Window

This is only available when a project is being debugged. It is a very powerful feature that allows the user to evaluate expressions and to display the runtime object content.

The dialog box has an expression entry at the top. At the center of the dialog box there is the object viewer, which displays the result of the expression evaluation in a tree view. If the user enters the name of a function variable that contains an object as an expression, it will be displayed in the object viewer. The object tree root node represents the object in the variable. There will be a child node per object contained in the root object, and each of these nodes will have a child node per object it contains and so on. This allows the user to inspect the object completely.

The object viewer has two sides. The left side is where the tree nodes are located, and the right side is where the value of the object, if applicable, is displayed. Typically the objects with values are the basic types: boolean, integer, string, number and blob. But each object has an internal value that can or cannot be used. For example, the standard object "date" has an integer as internal value to store the date value.

On each node the name of the variable or type property that holds an object, the type of the variable or type property holder and an internal ID of the current object is displayed. If there are two nodes with the same internal ID in a tree, it means that they refertothe same physical object. All the objects have a child type object. This type object transports information about the structure of the parent object. So the value of the type object will be the type the parent object has at runtime. For example, if the type of a variable is a tag type "type(MyType)" then the object held in the variable can or cannot be a "MyType" object. It is displayed in the child type object.

If an expression is not a variable or a type propery, result will be displayed in the object viewer as a tree with just one node and of another color.

The expressions can contain variables and type property names with boolean, integer and string constants. There are many operators that can be used in an expression, so we can evaluate very

complex ones at runtime and retrieve interesting pieces of information. (The expression rules are described in the the section called “Expression evaluation help”)

There is an edit box at the botton of the dialog box. When a node of the object tree that holds a basic object is selected, the value will be displayed in the box, and the user can modify it. After a value modification, the user has to press the "Set Value" button if he wants to update the object tree with this new value. After editing and changing several values in the object tree, the user must press the "Save new values" button to save all the changes in the physical objects.

When editing a blob value, a new window appears at the bottom of the dialog. This new window will display the ASCII translation of the binary content of the blob. This is quite useful if the blob contains ASCII information, e.g. ASCII text.

Thread Manager

The *Thread Manager* is the place to modify the running status of a thread whilst debugging a SIMPOL program.

The debugger enumerates the threads sequentially as they are created by the program, with the first one created being known as "Thread 1"

The *Thread Manager* displays the running status of all the threads in the program and the functions that they are executing at the time the thread manager is opened. The user can also suspend or resume any thread and change the debugger focus to another thread. This means that Step Into, Step Out, Run to Cursor etc. will affect this new thread.

Revision #1
Created 20 June 2021 17:35:34 by Nikolaus Zolnhofer
Updated 20 June 2021 17:41:44 by Nikolaus Zolnhofer