SUPERBASE NG IDE

Quick Start Manual

JUNE 21, 2018

SUPERBASE SOFTWARE LIMITED

Contents

Introduction

This book is intended to provide a quick start guide for using the new Superbase NG IDE. Although in the process some SIMPOL programming is covered, that is not the primary purpose of this book. Other books are provided that cover SIMPOL programming.

The parts that follow will describe the use of the IDE for programming standard applications that are run at the console and from within the IDE and also cover the creation of web server applications together with information about downloading and installing a web server. We will use AMPPS as it is an easy out of the box Apache server. Although it is not necessary to use Apache to deploy the resulting programs, Apache is an excellent web server and the debugging of web applications in the IDE is currently only supported using CGI, not using the ISAPI or Fast-CGI methodologies.

A large number of screen shots are used to explain the material and a step-by-step approach is taken with respect to the actual creation of the programs. Please take the time to work through the tutorials for the various programming styles. It will be time well spent.

Getting Started with the Superbase NG IDE

When the Superbase NG Integrated Development Environment (IDE) first opens you will see a picture similar to the one below:

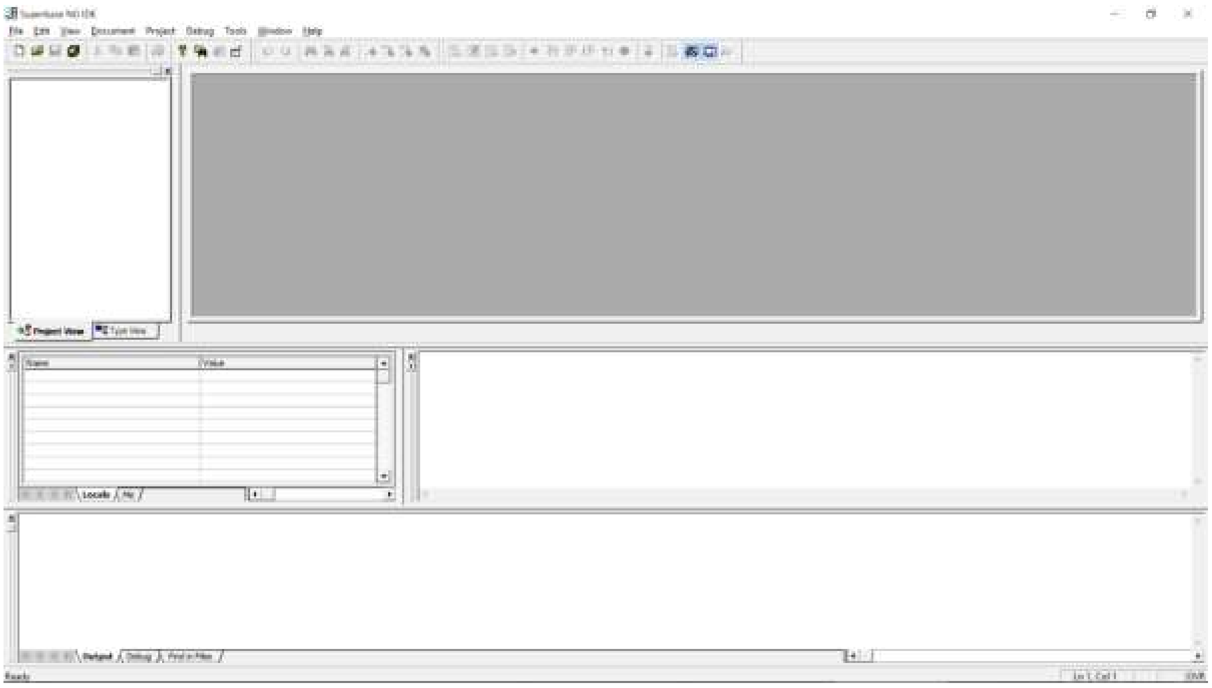


Figure 1 Initial State of the Superbase NG IDE

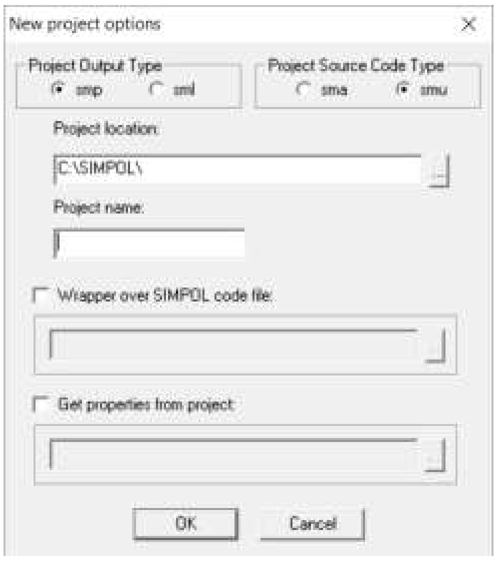
Let's take a look around the interface and examine the various features. At the very top is the menu, and directly below that the toolbars. Although the menu is fixed, the toolbars are dockable and can be placed in various locations in the main window frame. In the upper left corner is the project window, which provides two different views of the project, the file view and the type view. To the right of that is the editor area which is an MDI area. Multiple source code and other document windows can be opened in this area. If they are maximized they can still easily be reached by clicking on the tab at the bottom of the area that represents the desired window (this will be seen later). Below that is the output panel, which contains the various output areas as separate tab regions within the same area. This panel is where the results of a compilation or the running of a program can be found. In the middle there are the Watch and Call Stack panels. On the left is the variable and object Watch panel. Unlike many common watch windows, this one always shows all of the variables for the current function once the variables have been declared in the flow of the program. The Me tab is provided so that the object passed to an event procedure can be examined easily (it does not need to be called `Me` in the function declaration). The last panel we see is the Call Stack panel. This panel shows the current state of the program in reverse order. In other words, if the program started in the `main()` function, and then called the `init()` function, which called the `init_databases()` function and we stopped execution within the last function then the Call Stack panel would show the three functions in the reverse order that we called them. Also, the Watch panel would contain the current state of the variables in the `init_databases()` function. If we then clicked on one of the earlier functions, the Watch window would then show us the state of the variables in *that* function.

The IDE is a very flexible environment and it is easily adapted to look the way any one person wishes to work. Panels can be turned on or off, they can be resized and rearranged. The toolbars can also be moved, undocked an arranged in a different manner that may better suit the user.

Now that we have had a look around we will, in the next section, build our first project, a very basic program that is designed to teach us more about the development environment.

Creating Our First Project

To create a new Superbase NG project, select the New Project from the File menu. At that point, the following window will be shown:



New project options

Project Output Type: [✓ smp] sml

Project Source Code Type: sma [✓ smu]

Project location: C:\SIMPOL\

Project name:

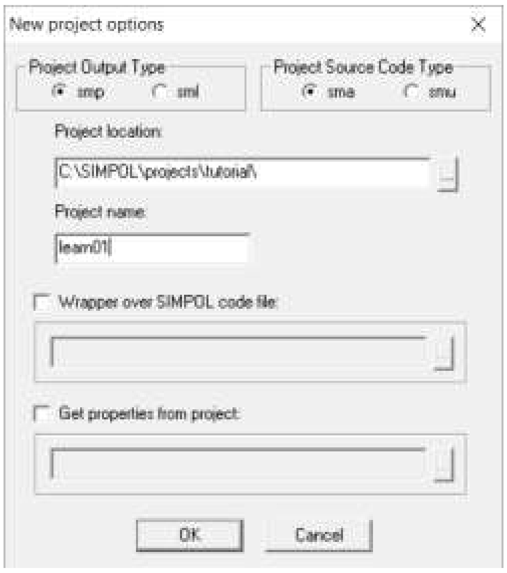
Wrapper over SIMPOL code file:

Get properties from project:

Figure 2 The New Project Options window

In this window we define the project name, project output type (`sml` or `smp`), source code file type (`sma` or `smu`), where the project should be located and what it should be called. The options in the lower half of the window are advanced options that we won't cover at this time. In this example we will select the smp and sma options. The sml option is for producing compiled libraries of types and/or functions for use by yourself and others. The smu extension is for creating Unicode source files rather than ANSI ones. In the following examples we will only use the ANSI source file types. Unicode can be very useful when working with characters from multiple code pages, such as mixing Greek and western European languages, but is not available when working with Windows 9x and is therefore not appropriate as the standard type for source files when supporting all platforms.

For the purposes of our first example, select the smp and sma options. Now click on the … button next to the Project location box. Beneath the `Projects` directory if no directory called "tutorial" is present then please create one using the Make New Folder. Once a `tutorial` directory exists below the `Projects` directory, select it and click on the OK button. In the Project name box enter **`learn01`**. Do not add any extension to this, the resulting program will be called `learn01.smp`, the main source file `learn01.sma` and the project directory `learn01`. See the picture below for details:



New project options

Project Output Type: [✓ smp] sml

Project Source Code Type: [✓ sma] smu

Project location: C:\SIMPOL\projects\tutorial\

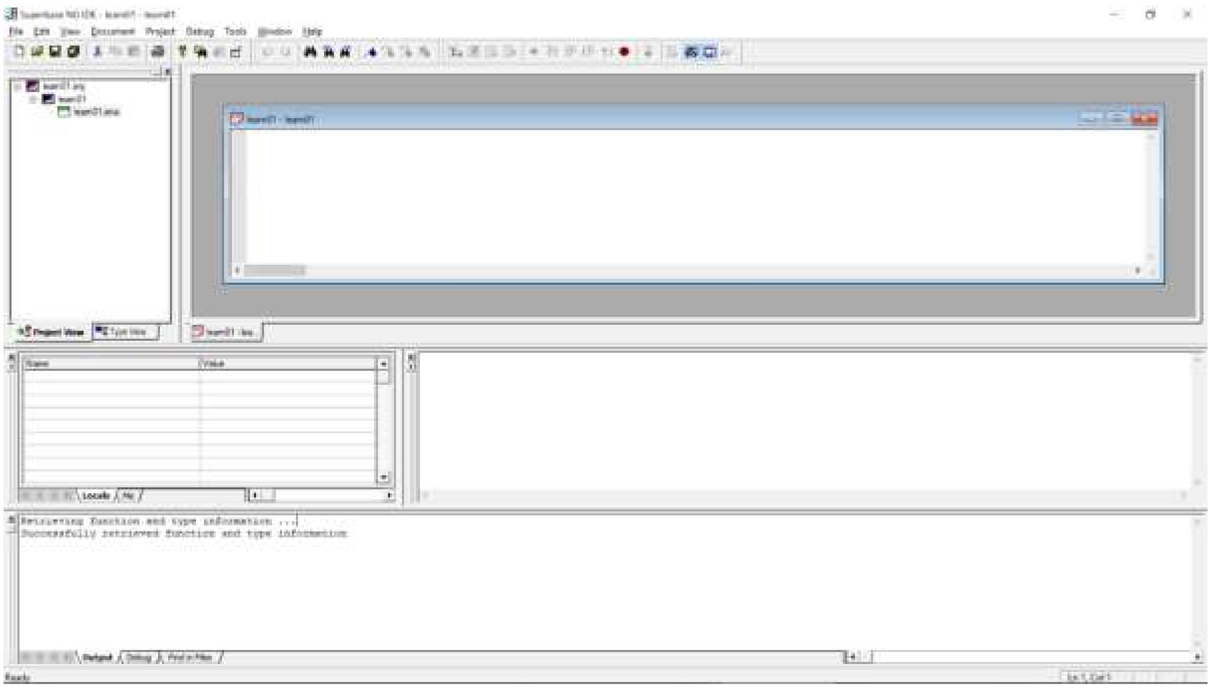
Project name: learn01

Wrapper over SIMPOL code file: [ ]

Get properties from project: [ ]

Figure 3 The New Project options window with the correct input for learn01

Clicking on the OK button will create a new project with the name "learn01". The project will be opened, the main source file will be created, and the result will look something like the picture below:



```
Retrieving function and type information ...
Successfully retrieved function and type information
```

Figure 4 The Superbase NG IDE with the learn01 project created

Now that we have a project we can start writing the program code.

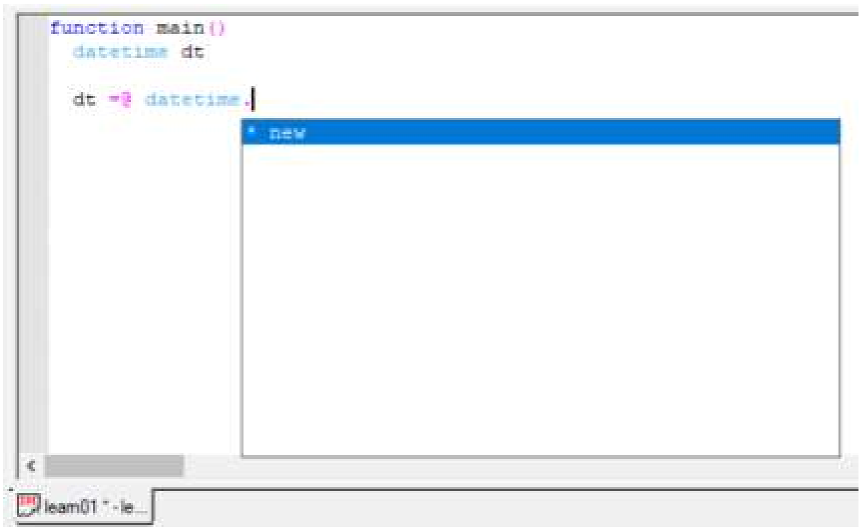
Writing Our First Program

The first program will be a very easy one with very little real purpose beyond demonstrating various capabilities of the development environment. Although the "Hello World" program is quite traditional, for this example we have chosen to create a program that outputs the current date and time. This has the necessary flexibility that is required for our demonstration. The source code in its entirety is shown below:

```
function main()
  datetime dt

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = .tostr(dt.year(), 10) + "/" + .tostr(dt.month(), 10) + \
      "/" + .tostr(dt.dayinmonth(), 10) + "  " + \
      .tostr(dt.hours(), 10) + ":" + .tostr(dt.minutes(), 10) + \
      ":" + .tostr(dt.seconds, 10)

end function s
```

Please type the program in, don't copy it from this document. The process of entering the source code will demonstrate a number of the features that we will discuss as we continue.

As you type in the first part of the code, as shown in the following picture, a number of things may occur to you. First, the various words and punctuation in the program appear in different colours. Color-coding of the source code is quite common today and assists the reader in immediately being able to focus on the parts of the program that are of interest as well as visually pointing out when things may have been done incorrectly. Which colors are used for what parts of the programming language are user-configurable. By default, language keywords appear in blue, identifiers in black, strings in red, data types in cyan, operators in magenta, and comments in green.

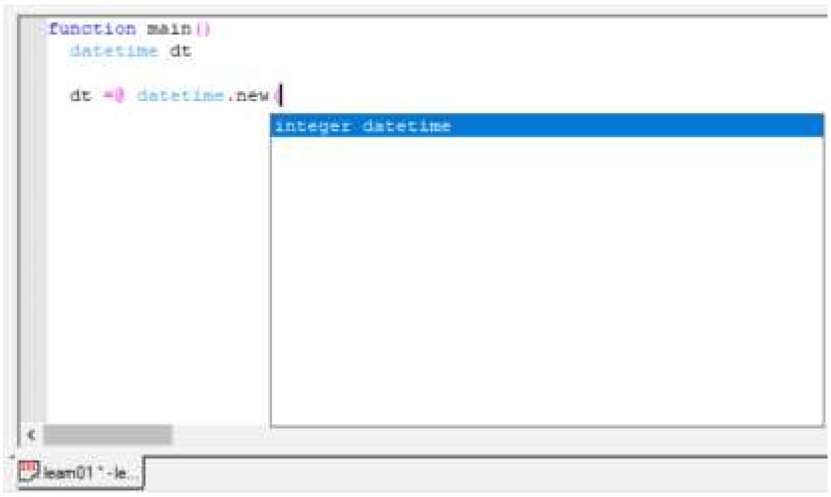


```
function main()
  datetime dt

  dt =@ datetime.
```

Figure 5 Showing the autcomplete functionality within SIMPOL

The picture above shows the inline programming help for the datetime type. Since the `new()` method is the only item in the list, it is already pre-selected. Press the **tab** key to autocomplete, the highlighted text will be entered at the current cursor position. This feature can greatly reduce the time it takes to write programs, as well as reduce the number of typing errors made.

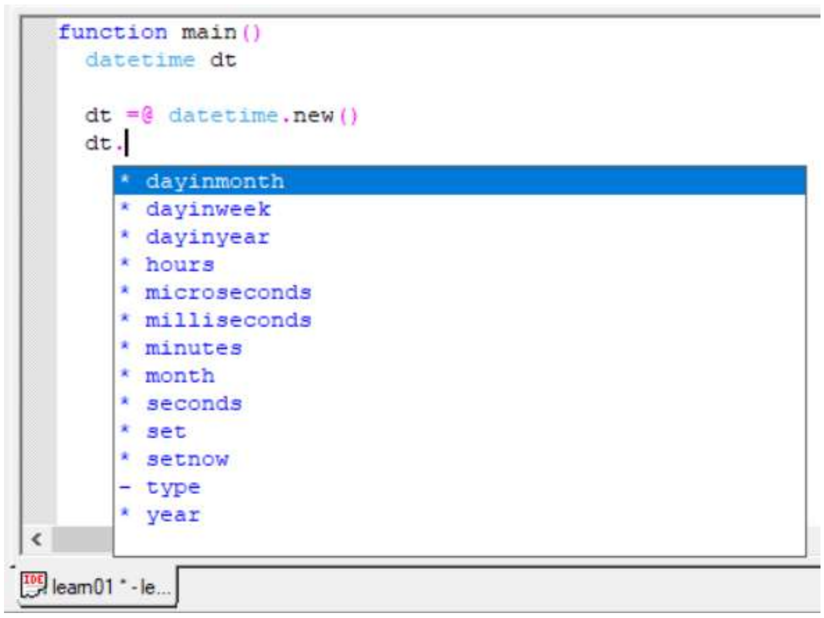
When typing the open parentheses next to a method name, such as `new`, the inline help will show the arguments for the call to the method, as seen below



```
function main()
  datetime dt

  dt =@ datetime.new(
```

```
integer datetime
```

Figure 7 The inline help for the new() method



```
function main()
  datetime dt

  dt =@ datetime.new()
  dt.
```

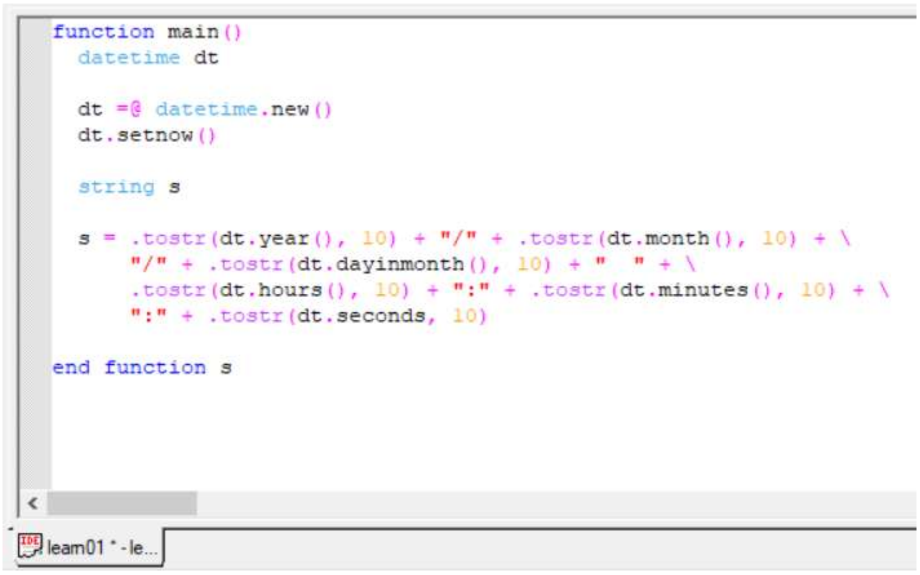
```
* dayinmonth
* dayinweek
* dayinyear
* hours
* microseconds
* milliseconds
* minutes
* month
* seconds
* set
* setnow
- type
* year
```

Figure 6 The inline help for the datetime object

Even when using a variable that is declared to be of the type datetime such as in our program the properties and methods of the object are shown by the inline help while typing the code. To select a different one than the first, just type the first one or two letters until the correct one is selected and then press the **tab** key to have the rest of the item entered at the cursor position.

Every component in SIMPOL has inline help. In some cases, like that of functions, the help only shows which parameter is current and needs to be filled out as well as information about its data type and possibly the parameter name and default value.

Now enter the remainder of the program as shown in the earlier source code excerpt. Once the entire source code has been entered, it should look like the following picture. At this point we are ready to build and test the project.



```
function main()
  datetime dt

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = .tostr(dt.year(), 10) + "/" + .tostr(dt.month(), 10) + \
      "/" + .tostr(dt.dayinmonth(), 10) + "  " + \
      .tostr(dt.hours(), 10) + ":" + .tostr(dt.minutes(), 10) + \
      ":" + .tostr(dt.seconds, 10)

end function s
```

Figure 8 The complete source code for the first project

Building and Testing Our First Program

The next step is to compile the program and then we can run it. To build the program, select the Build item from the Project menu as shown below. (Alternatively press **Ctrl+B**)

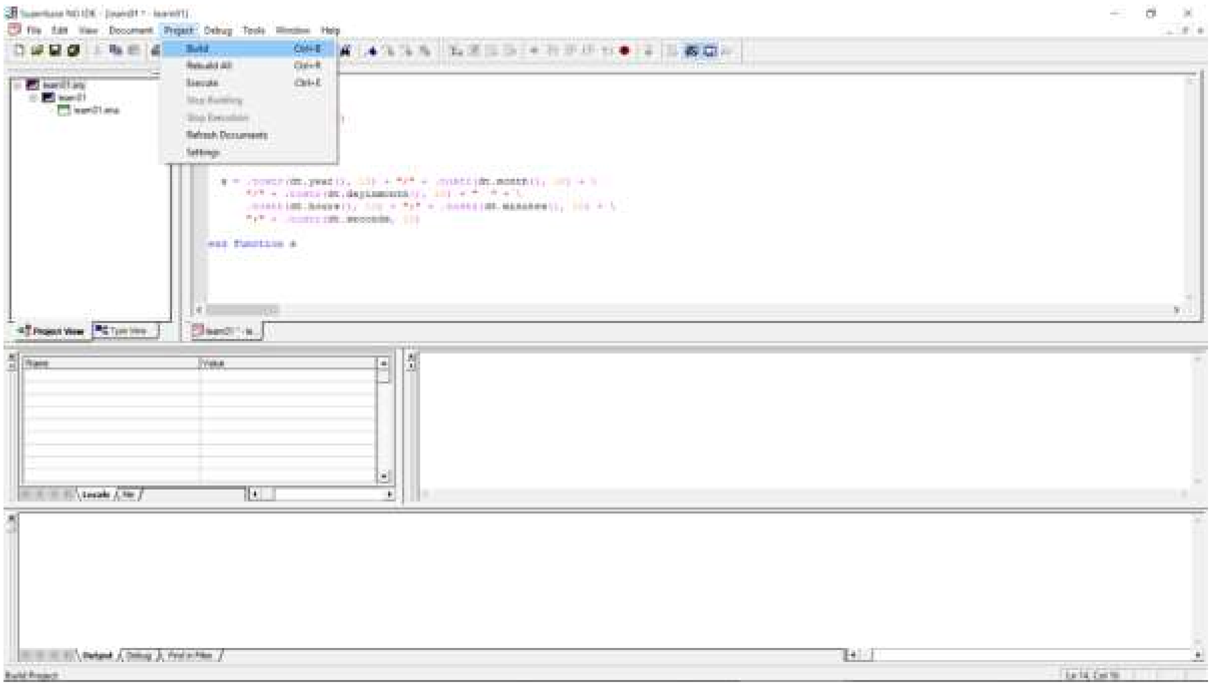
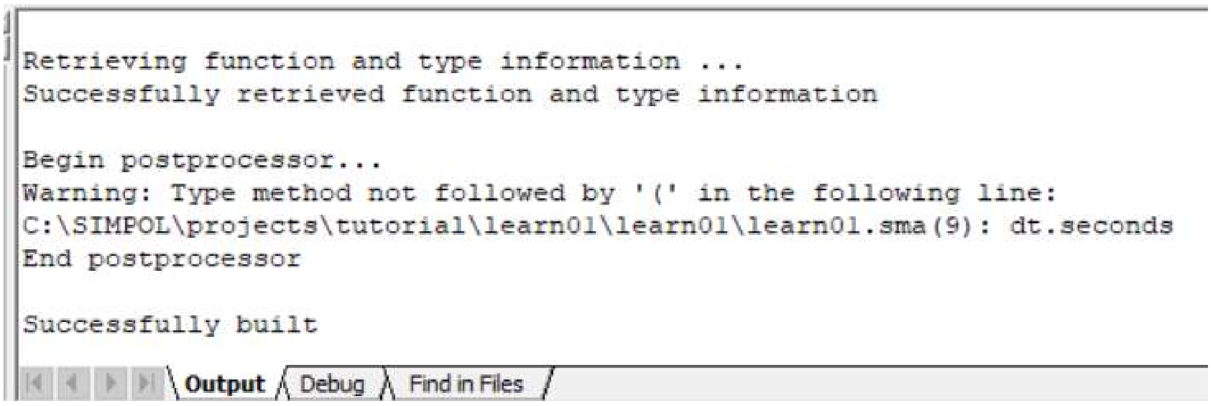


Figure 9 Building the first project

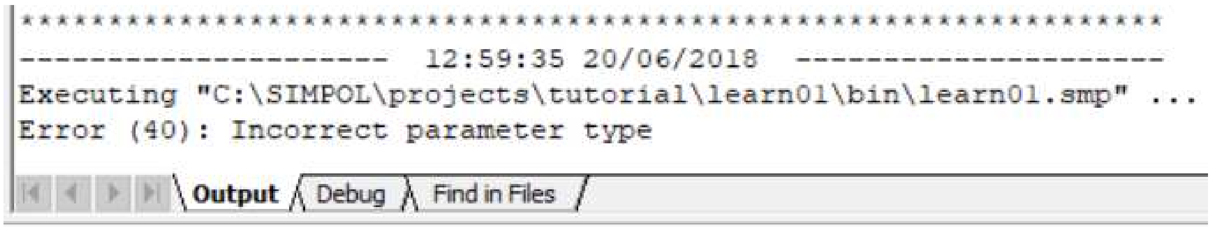
The compilation succeeded as can be seen in the output window:



```
Retrieving function and type information ...
Successfully retrieved function and type information

Begin postprocessor...
Warning: Type method not followed by '(' in the following line:
C:\SIMPOL\projects\tutorial\learn01\learn01\learn01.sma(9): dt.seconds
End postprocessor

Successfully built
```

Figure 10 The first project successfully compiled

For now we will ignore the warning from the post-processing code, but normally it is a good idea to try and deal with all of the warnings since they can otherwise result in runtime errors in the program. Now we can execute the program by either selecting the Execute item from the Project menu, or else by pressing **Ctrl+E**. As we can see from the picture below, the program had an error while executing.



```
******************************************************************
--------------------  12:59:35 20/06/2018  --------------------
Executing "C:\SIMPOL\projects\tutorial\learn01\bin\learn01.smp" ...
Error (40): Incorrect parameter type
```

Figure 11 The first project fails with an error

Now that we have had an error, it is time to start up the debugger. Select the Start Debugging item from the Debug menu. This can also be accomplished by pressing the **F4** key.

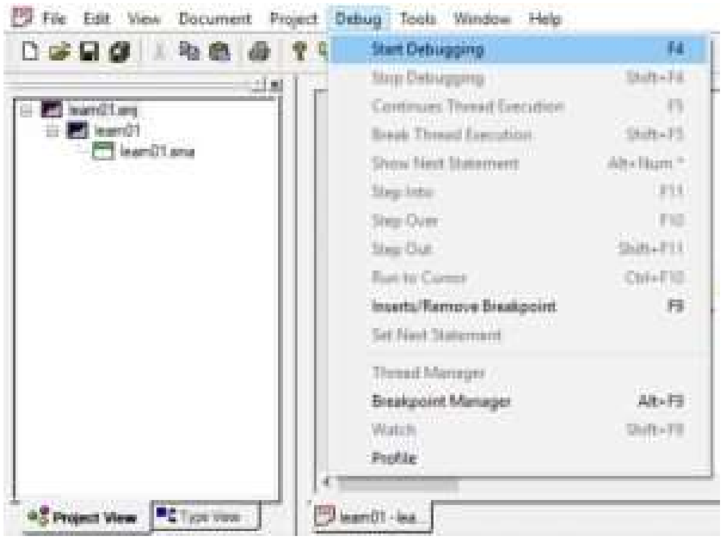
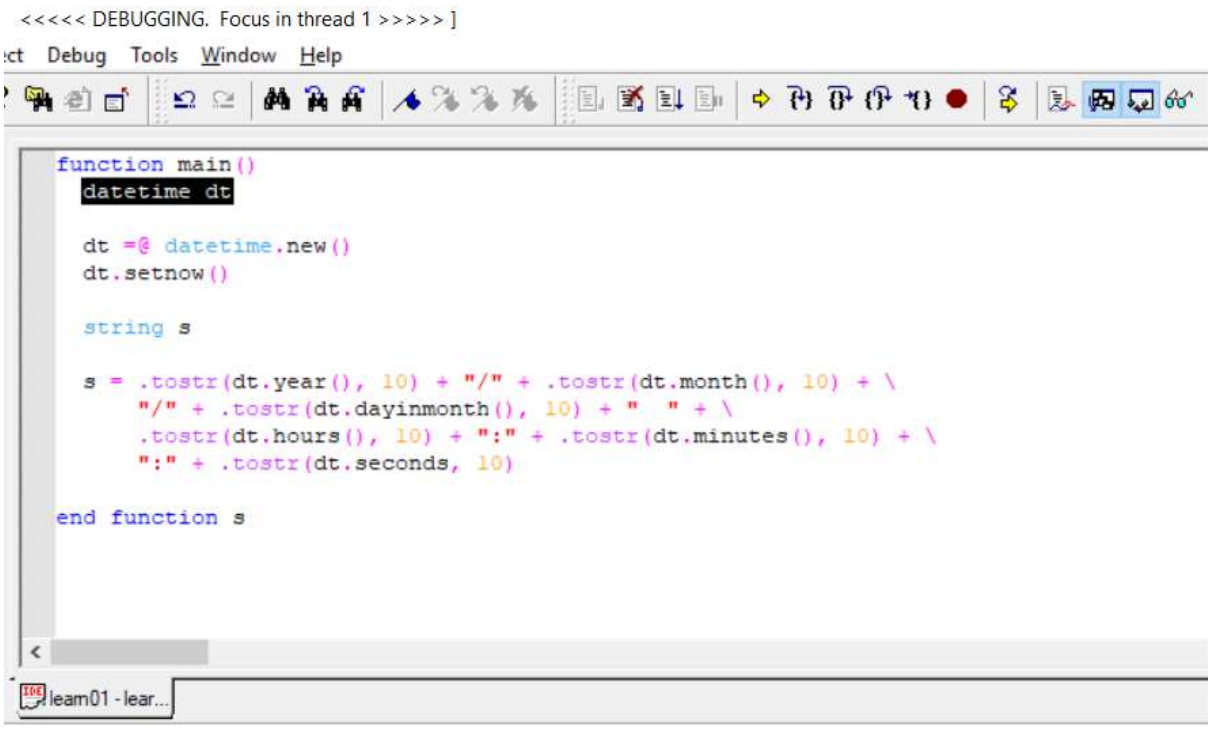


Figure 12 Starting the debugger

When the debugger starts, it first checks to see if the program has changed since it was last compiled and, if necessary, saves the project (if that is one of the settings) and recompiles the project. Then it initializes SIMPOL, loads and starts the program, and breaks execution on the first statement following the declaration of the `main()` function, as shown below:

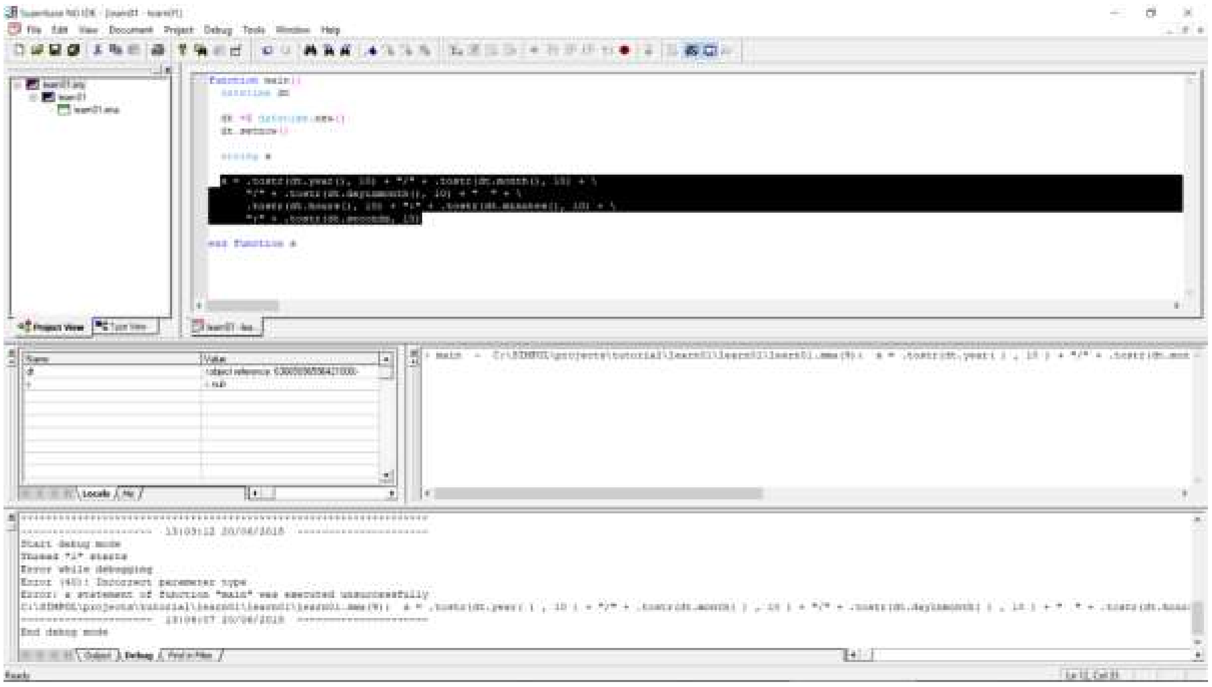


<<<<< DEBUGGING. Focus in thread 1 >>>>>]
ct Debug Tools Window Help
learn01 - lear...

```
function main()
  datetime dt

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = .tostr(dt.year(), 10) + "/" + .tostr(dt.month(), 10) + \
      "/" + .tostr(dt.dayinmonth(), 10) + "  " + \
      .tostr(dt.hours(), 10) + ":" + .tostr(dt.minutes(), 10) + \
      ":" + .tostr(dt.seconds, 10)

end function s
```

Figure 13 The debugger stopped on the first statement

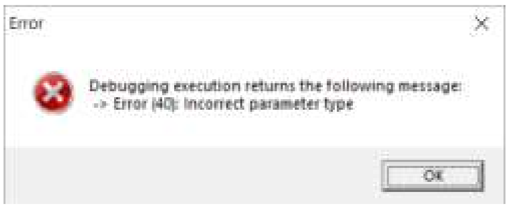
To single step through the code, press **F10**. We will eventually get to the line shown in the picture below:



```
-------------------------  13:03:12 20/08/2018  -------------------------
Start debug mode
Thread "1" starts
Error while debugging
Error (40): Incorrect parameter type
Error: a statement of function "main" was executed unsuccessfully
C:\SIMPOL\projects\tutorial\learn01\learn01\learn01.sma(9):  s = .tostr(dt.year( ), 10 ) + "/" + .tostr(dt.month( ), 10 ) + "/" + .tostr(dt.dayinmonth( ), 10 ) + "  " + .tostr(dt.hour
-------------------------  13:08:07 20/08/2018  -------------------------
End debug mode
```

Figure 14 The debugger stopped on the last statement inside the function

Pressing the **F10** key once more will result in a pop-up displaying the error message, it should look like this:



```
Error
Debugging execution returns the following message:
-> Error (40): Incorrect parameter type
```

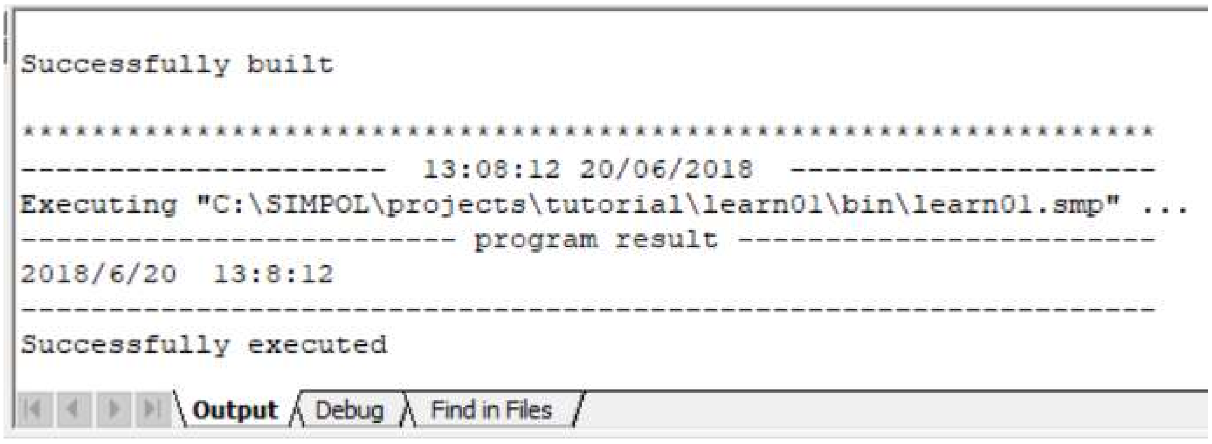
Figure 15 The error message pop-up

The error number is 40, "Incorrect parameter type". If we take a closer look at the source code, we can see that the first parameter to the `.tostr()` in the final segment of the last statement is `dt.seconds`. The error here is the missing parentheses, since `seconds()` is a method of the datetime type, not a property. Let's now change the source code to the correct syntax.

This is done by adding some brackets after `dt.seconds` the line should now look like this:

```
s = .tostr(dt.year(), 10) + "/" + .tostr(dt.month(), 10) +"/" +
.tostr(dt.dayinmonth(), 10) + \
"  " +.tostr(dt.hours(), 10) + ":" + .tostr(dt.minutes(), 10) +":" +
.tostr(dt.seconds(), 10)
```

We can now rebuild (**Ctrl**+**B**) and execute (**Ctrl**+**E**) the program. This time the program runs successfully without errors:



```
Successfully built

*************************************************************************
----------------------  13:08:12 20/06/2018  ----------------------
Executing "C:\SIMPOL\projects\tutorial\learn01\bin\learn01.smp" ...
------------------------- program result -------------------------
2018/6/20  13:8:12
-------------------------------------------------------------------
Successfully executed
```

Figure 16 The program has run successfully

The date and time displayed will of course not be the same

Upon careful examination of the output from the program, however, we can already see that there is still some improvement that can be made over the current version. The program output the string `2018/6/20  13:8:12`. Although the date might be considered acceptable, the time is certainly not going to be acceptable in the current format by most people. In the next section we will improve the current program by making incremental improvements and by making use of supplied library functionality that itself was written in SIMPOL.

Making Incremental Improvements

The reason why the initial version of our program, though functional, was not acceptable is that the output was not formatted in a way the user may expect or desire. Part of the reason lies in the fact that to start with, we used the SIMPOL intrinsic function `.tostr()`. Although this function is quite useful, it is a fairly low-level function and does not provide a wide degree of flexibility when formatting the result. For that reason, early in the development cycle of SIMPOL, additional functions were written using SIMPOL itself to provide that sort of functionality.

There is currently a large and ever-growing library of pre-designed functionality supplied with SIMPOL and in most cases the fully commented source code of the library is also provided. Pre-compiled libraries are located by default in the `lib` subdirectory of the place where Superbase NG was installed. Projects are normally located in the `Projects` directory also located directly below the root directory of the installation. The source code for the various supplied libraries can be found in the `Libs` directory located directly below the `Projects` directory.

In order to improve the output of the program, we can use the `STR()` function found in the `STR.sml` library file. This function includes the ability to format strings in exactly the same ways as those supported by the previous Superbase product, except currently for a lack of support for scientific notation, which will eventually also be supported. To access the functionality in this library, we first need to add it to the project. Select the Settings item from the Project menu.

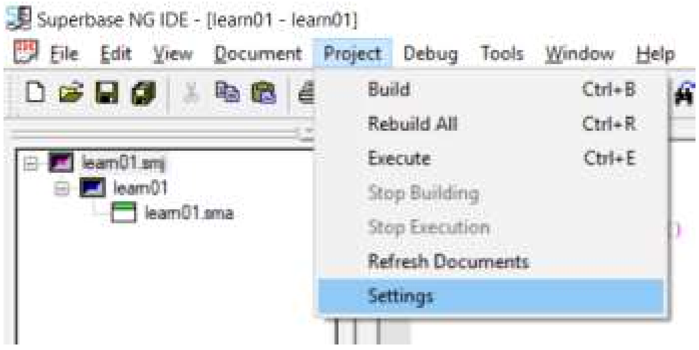
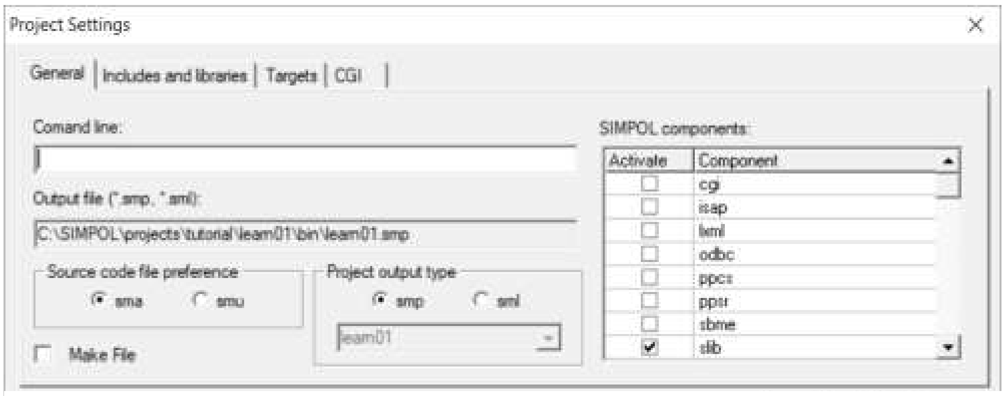


Figure 17 Opening the Project Settings window via the menu

This will display the Project Settings window.


Project Settings

General | Includes and libraries | Targets | CGI

Comand line:

Output file (*.smp, *.sml): C:\SIMPOL\projects\tutorial\learn01\bin\learn01.smp

Source code file preference: (• sma) (smu)

Project output type: (• smp) (sml)

learn01

☐ Make File

SIMPOL components:

Activate	Component
☐	cgi
☐	isap
☐	lxml
☐	odbc
☐	ppcs
☐	ppsr
☐	sbme
✓	slib

Figure 18 The Project Settings window

This window is extremely important for creating powerful and successful applications using SIMPOL. The initial tab allows the setting of the source code file preference, assigning of command line parameters when running and debugging in the IDE, and also provides a method of selecting the SIMPOL components required by the project. If a component is required but not selected, then it will not be available at runtime nor will the inline help assistance work for the associated types and functions.

The second tab provides a place to define two important areas, on the left is the place that the include directories are added (where the compiler will look for included source code files during compilation) and on the right is the list of pre-compiled SIMPOL modules (`*.sml`'s) that are to be added to the project.

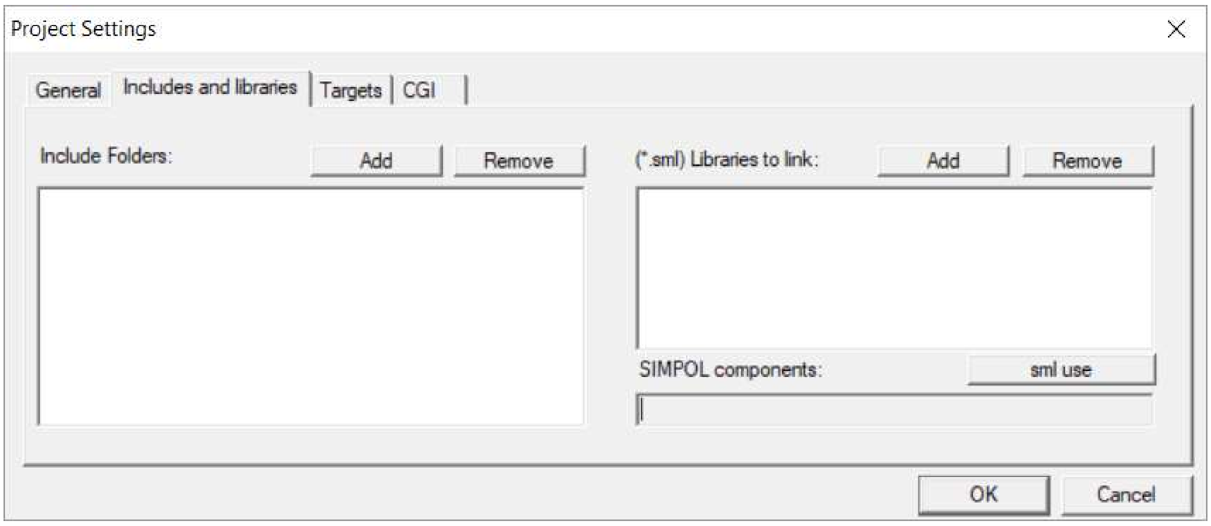
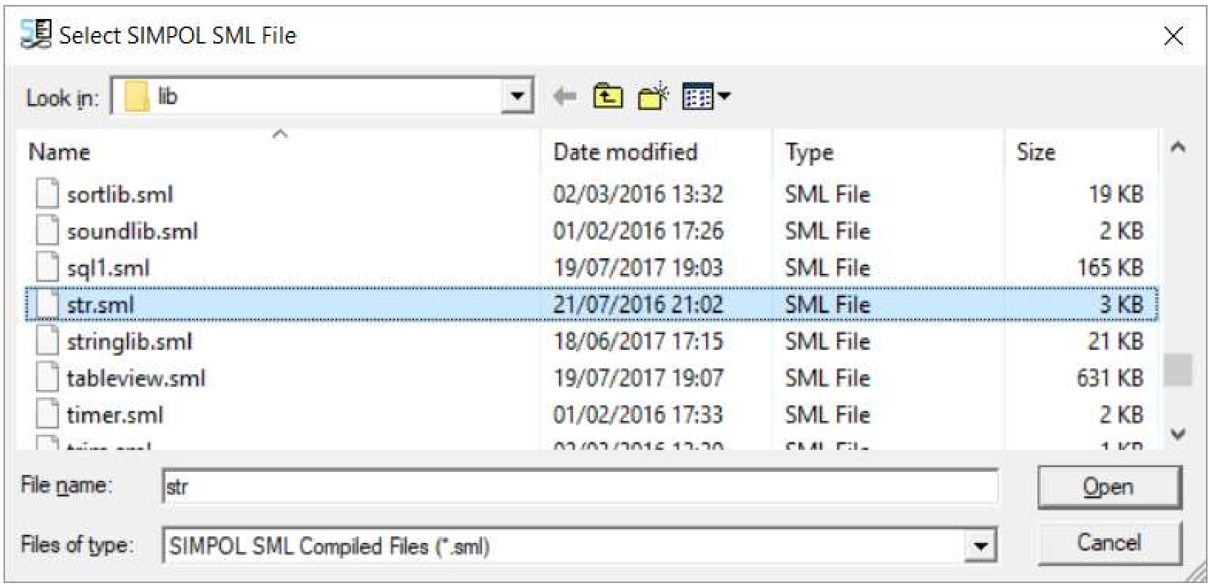


Figure 19 The Project Settings Includes and Libraries tab

Click on the Add button on the right side of the window and from the resulting file selection window, go into the `lib` directory and select the file `STR.sml`.



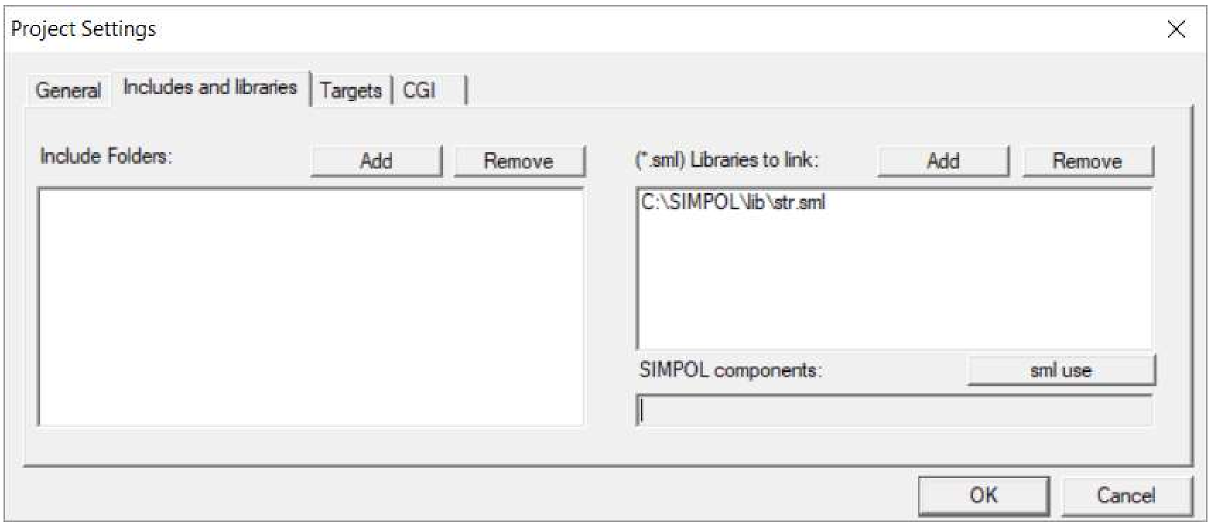
Name	Date modified	Type	Size
sortlib.sml	02/03/2016 13:32	SML File	19 KB
soundlib.sml	01/02/2016 17:26	SML File	2 KB
sql1.sml	19/07/2017 19:03	SML File	165 KB
str.sml	21/07/2016 21:02	SML File	3 KB
stringlib.sml	18/06/2017 17:15	SML File	21 KB
tableview.sml	19/07/2017 19:07	SML File	631 KB
timer.sml	01/02/2016 17:33	SML File	2 KB

File name: str

Files of type: SIMPOL SML Compiled Files (*.sml)

Figure 20 The file selection window for STR.sml

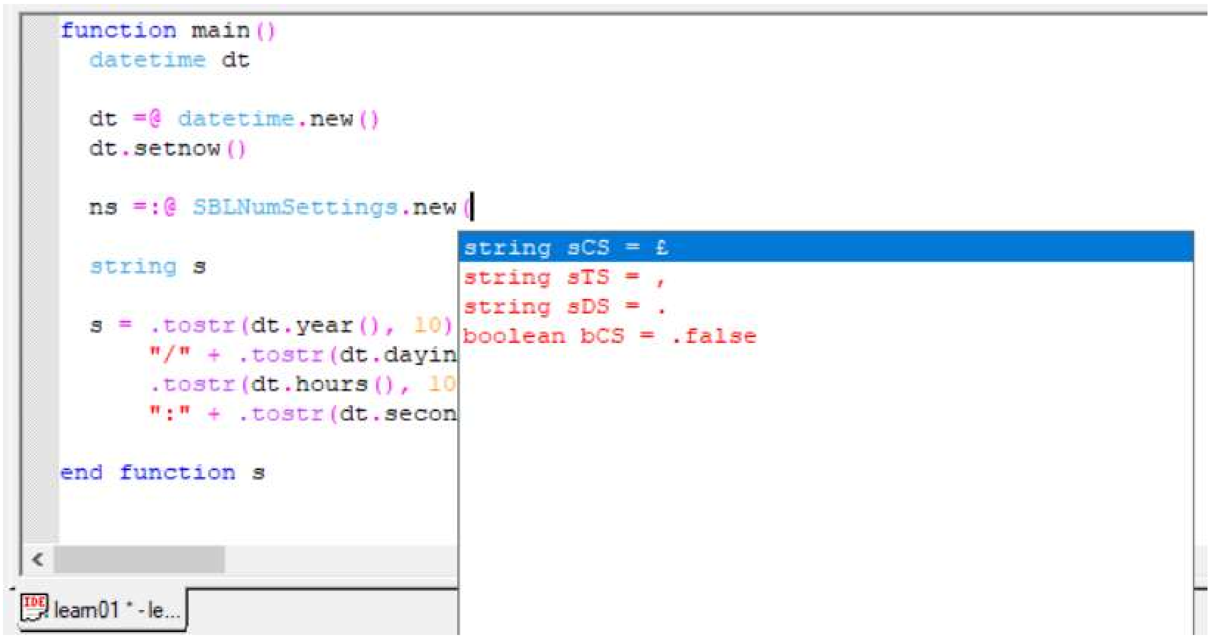
The project settings menu should now look like this



C:\SIMPOL\lib\str.sml

Figure 21 Project Settings Includes and Libraries tab with STR.sml

Once the library has been added to the project, we can add a declaration for the type SBLNumSettings, which is necessary in the SIMPOL version of this function because unlike in SBL and other BASIC derivatives, there are no pre-defined global entities such as `Superbase.NumericFormat`. As can be seen from the following picture, the inline help also supports user-defined objects and functions. In this case the `new`() has been implemented in such a way as to allow default values for the object which the user can override by passing in other values.



```
function main()
  datetime dt

  dt =@ datetime.new()
  dt.setnow()

  ns =:@ SBLNumSettings.new(

  string s

  s = .tostr(dt.year(), 10)
      "/" + .tostr(dt.dayin
      .tostr(dt.hours(), 10
      ":" + .tostr(dt.secon

end function s
```

```
string sCS = £
string sTS = ,
string sDS = .
boolean bCS = .false
```

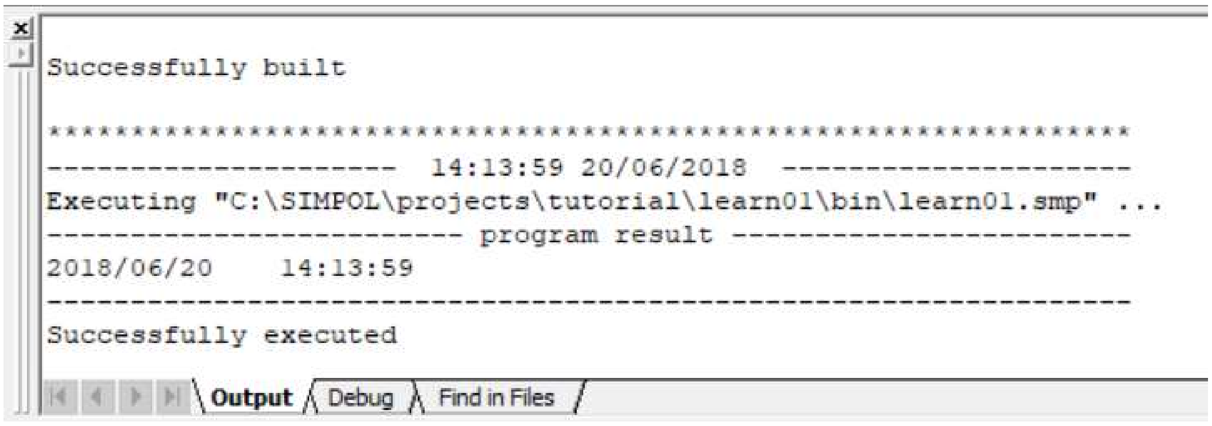
Figure 22 The inline help for the SBLNumSettings type

Continue modifying the code until it matches the source code below. We will replace nearly all instances of the function `.tostr()` with the function `STR()`. This will give greater flexibility when formatting our numbers.

```
function main()
  datetime dt
  SBLNumSettings ns

  ns =@ SBLNumSettings.new()

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = .tostr(dt.year(), 10) + "/" + STR(dt.month(), "00") + "/" + \
      STR(dt.dayinmonth(), "00", ns) + "  " + STR(dt.hours(), "00", ns) + \
      ":" + STR(dt.minutes(), "00", ns) + ":" + STR(dt.seconds(), "00", ns)

end function s
```

Notice the use of the forward slash (\) in order to carry on a declaration over a line

Now we can rebuild the project by pressing **Ctrl**+**B**. Assuming that no typing mistakes were made and that it builds successfully, pressing **Ctrl**+**E** should successfully run the program and show the results in the output window, which should look something like the following (obviously the actual date and time will differ).



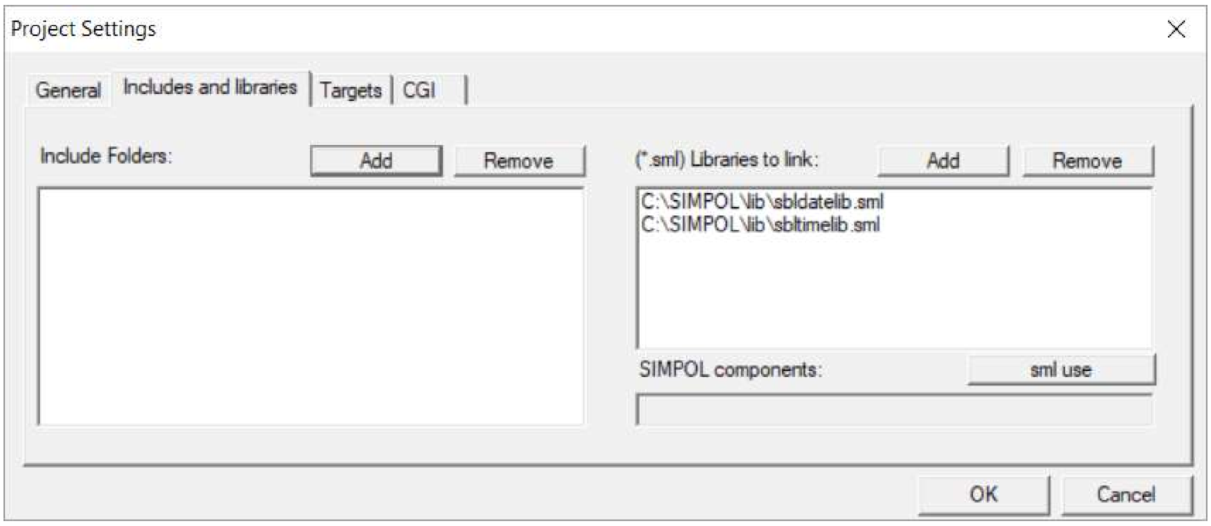
```
Successfully built

******************************************************************
---------------------  14:13:59 20/06/2018  ---------------------
Executing "C:\SIMPOL\projects\tutorial\learn01\bin\learn01.smp" ...
------------------------- program result -------------------------
2018/06/20    14:13:59
------------------------------------------------------------------
Successfully executed
```

Figure 23 The output from the modified program

This time around the result looks much more reasonable than the earlier version. This solution still leaves some open issues, such as dealing with date formats that use the name or the abbreviation of the month and the am/pm style of time. The solution to this is to use more appropriate functions for the formatting of the date and time. As it turns out, just as there is a `STR.sml` there is also a library called `SBLDateLib.sml` and another called `SBLTimeLib.sml`, both of which were written in SIMPOL and for which the source code is provided. These libraries are intended to be directly compatible with the older SBL functionality and they contain functions that are in all capital letters, such as `DATESTR()`, `MONTHSTR()`, `TIMESTR`(), and others. As the development of SIMPOL progressed we created numerous libraries that reproduce the functionality from SBL as well as producing more modern versions of some functions. For example, one of the functions included is the

LTRIM() function. This function is a drop-in replacement for the SBL function of the same name. There is also a function supplied called ltrim(). This function is a bit more sophisticated than the SBL version, in that it not only trims spaces, it also trims tab characters and can be passed a string parameter to optionally trim any character contained within that string so that the user can choose which characters should be trimmable.

Let's make some final improvements to the program. Reopen the Project Settings window via the menu and in the Includes and libraries tab remove the STR.sml and select instead the SBLDateLib.sml and SBLTimeLib.sml from the lib directory. The Project Settings window should look like this:



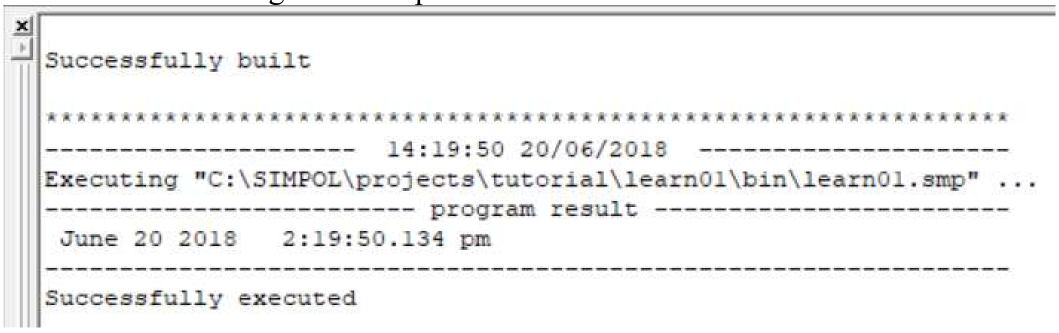
C:\SIMPOL\lib\sbldatelib.sml
C:\SIMPOL\lib\sbltimelib.sml

Figure 24 Project Setting Includes and Libraries tab for new version of learn01

Now modify the source code to make use of these new libraries as follows:

```
function main()
  datetime dt
  SBLlocaledateinfo ldiLocale
  integer iMicrosecondsinaday

  ldiLocale =@ SBLlocaledateinfo.new()
  iMicrosecondsinaday = 60 * 60 * 24 * 1000000

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = DATESTR(date.new(dt/iMicrosecondsinaday), "mmmm dd, yyyy", \
              ldiLocale) + "   " + \
      TIMESTR(time.new(dt mod iMicrosecondsinaday), "hh:mm:ss.s am")
end function s
```

This time after building and executing the program we can see that we are now able to finely control the formatting of the output.



```
Successfully built

************************************************************
---------------------  14:19:50 20/06/2018  ---------------------
Executing "C:\SIMPOL\projects\tutorial\learn01\bin\learn01.smp" ...
------------------------ program result ------------------------
 June 20 2018   2:19:50.134 pm
----------------------------------------------------------------
Successfully executed
```

Summary

In this part we have learned how to:

- Create a new project in the IDE
- Make use of the inline help
- Build and execute a project
- Debug a project
- Work with SIMPOL libraries (`*.sml`)

In the next part we will take our current project and learn how to modify it to output the results in a web page as a web server application.

Writing Web Server Programs With SIMPOL

In the previous part, we built our first basic program in SIMPOL and learned how to use the IDE to do various tasks. In this part, we will take that project and convert it into a web server application to output the same information to a web browser.

Converting Our Previous Project

As a first step, we can save our project from the first part as a new project. To do so, open the first project in the IDE and then select "Save Project As …" in the File menu

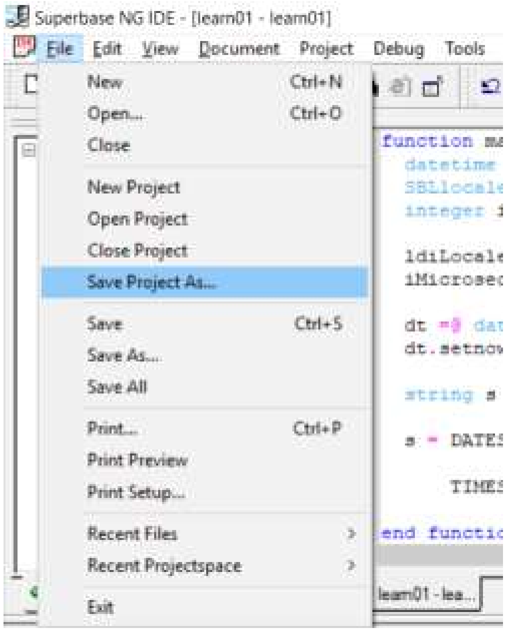
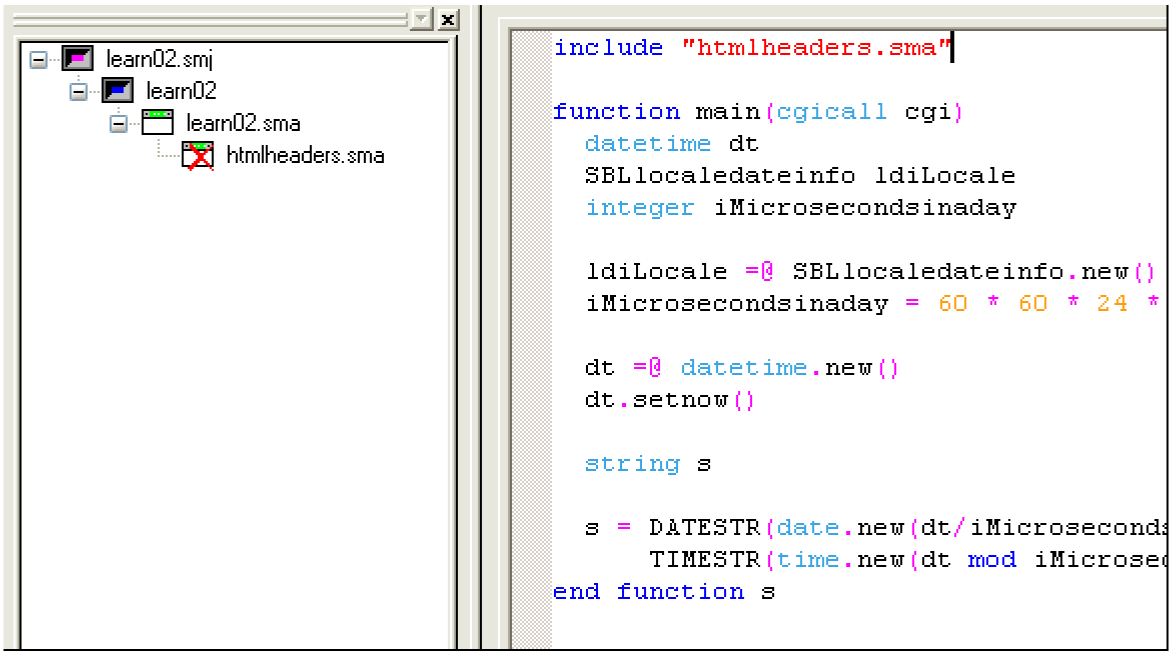


Figure 25 Saving the project with a different name

Then give the new project the name `learn02` and select the `tutorial` directory as the place to store it. This will create a new project called learn02 with a main source file called `learn02.sma`.



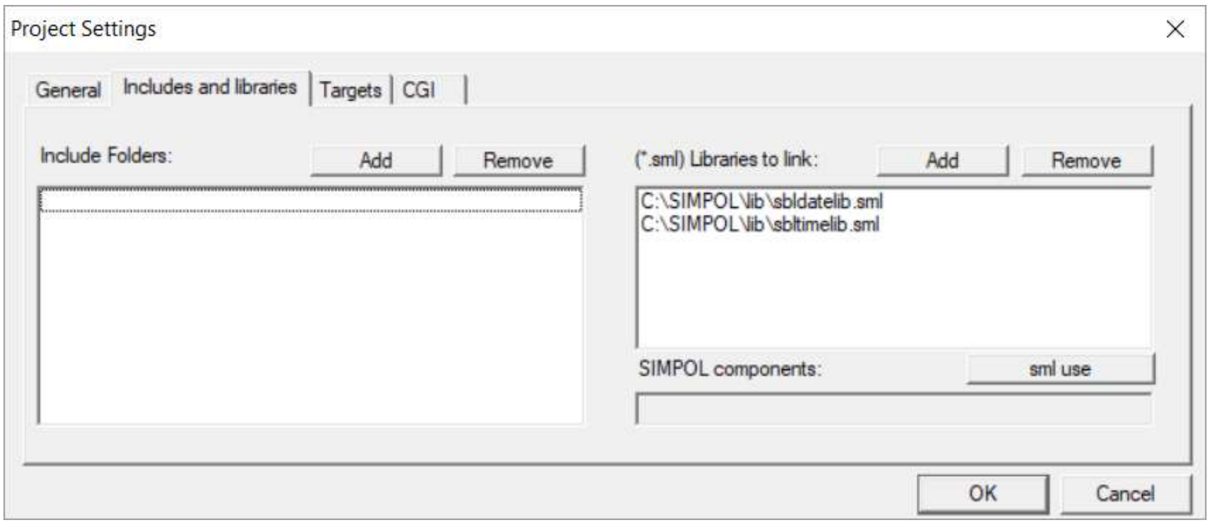
learn02.smj
learn02
learn02.sma
htmlheaders.sma

```
include "htmlheaders.sma"

function main(cgicall cgi)
  datetime dt
  SBLlocaledateinfo ldiLocale
  integer iMicrosecondsinaday

  ldiLocale =@ SBLlocaledateinfo.new()
  iMicrosecondsinaday = 60 * 60 * 24 *

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = DATESTR(date.new(dt/iMicroseconds
      TIMESTR(time.new(dt mod iMicrosec
end function s
```

Figure 26 The Project View for learn02

Now start modifying the source code. Add a `cgicall cgi` parameter inside the parentheses of the `main`() function. Then add the include statement at the top as shown below and press **Ctrl**+**S** to save the document. The result should look something like the following picture:

Note that the IDE has added a document as a child to the main document of the project but that the icon for the document has a red X through it. This is because at this point the project does not have any idea where to look for the include file. To fix this, let's open the Project Settings window again and switch to the Includes and libraries tab.



C:\SIMPOL\lib\sbldatelib.sml
C:\SIMPOL\lib\sbltimelib.sml

Figure 27 Adding an include path

Now click the Add button on the left side of the window and select the `include` directory as highlighted below:

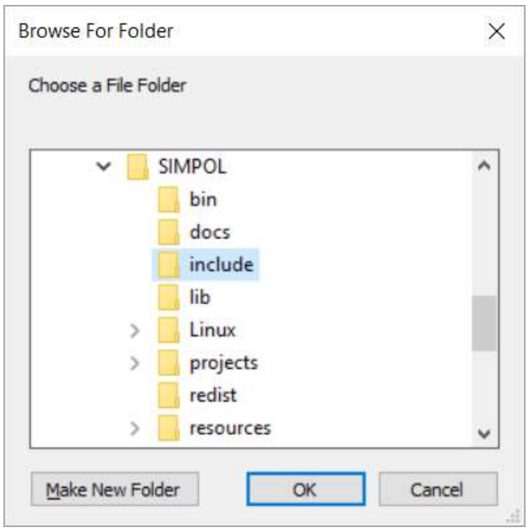
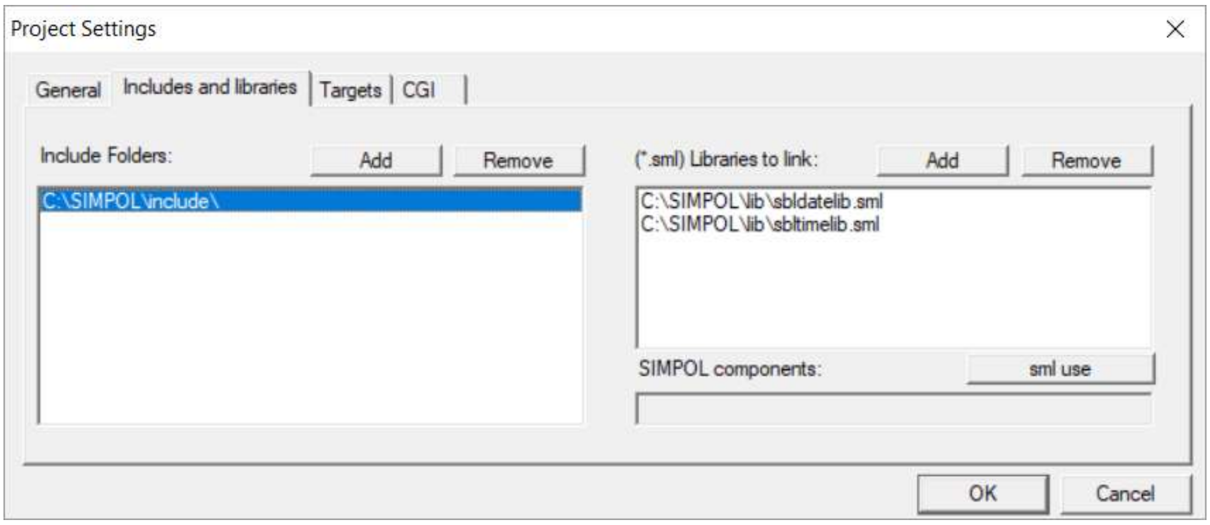


Figure 28 File browser showing include folder

Press OK, the window should then look like the one below



Include Folders:

C:\SIMPOL\include\

(*.sml) Libraries to link:

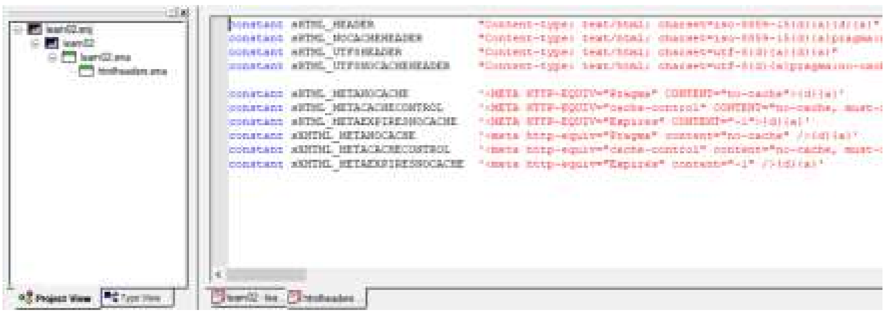
C:\SIMPOL\lib\sbldatelib.sml
C:\SIMPOL\lib\sbltimelib.sml

SIMPOL components:

Figure 29 The include path has been added

After clicking on OK the icon that previously was marked with an X is now back to normal.

Now double-click the included file and copy the constant value `sHTML_HEADER` to the clipboard so that you can paste it into the main program file. The constant value can be used as long as it has been defined prior to it's being used. Since the SIMPOL compiler is a single-pass compiler, that means that the file containing the constant needs to be included at the beginning of the program.



```
constant sHTML_HEADER                "Content-type: text/html; charset=iso-8859-15{d}{a}{d}{a}"
constant sHTML_NOCACHEHEADER         "Content-type: text/html; charset=iso-8859-15{d}{a}pragma:
constant sHTML_UTF8HEADER            "Content-type: text/html; charset=utf-8{d}{a}{d}{a}"
constant sHTML_UTF8NOCACHEHEADER     "Content-type: text/html; charset=utf-8{d}{a}pragma:no-cac

constant sHTML_METANOCACHE           '<META HTTP-EQUIV="Pragma" CONTENT="no-cache">{d}{a}'
constant sHTML_METACACHECONTROL      '<META HTTP-EQUIV="cache-control" CONTENT="no-cache, must-
constant sHTML_METAEXPIRESNOCACHE    '<META HTTP-EQUIV="Expires" CONTENT="-1">{d}{a}'
constant sXHTML_METANOCACHE          '<meta http-equiv="Pragma" content="no-cache" />{d}{a}'
constant sXHTML_METACACHECONTROL     '<meta http-equiv="cache-control" content="no-cache, must-
constant sXHTML_METAEXPIRESNOCACHE   '<meta http-equiv="Expires" content="-1" />{d}{a}'
```

Figure 30 Opening the included file

At this point we need to add the code that outputs the HTML page to the browser. Generally the second parameter to the `output()` method of the cgicall type will be set to 1, since currently most protocols require single-byte characters. The first thing that needs to be output is the header (unless you are using cookies, then they have to be first). After that the normal HTML code is output.

Now complete the code as shown below. The final argument following the `end function` statement is the empty string. This is because if it is not set to the empty string, the string representation of the value `.nul` will also be returned at the end of the HTML page.

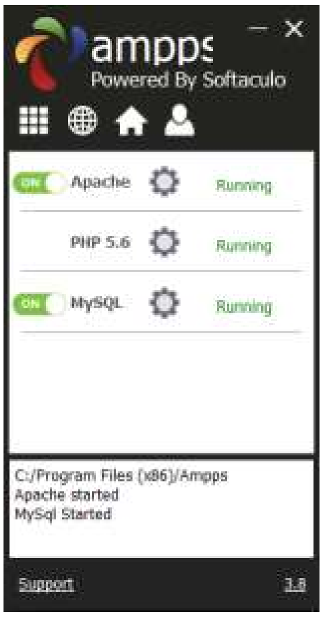
```
include "htmlheaders.sma"

function main(cgicall cgi)
  datetime dt
  SBLlocaledateinfo ldiLocale
  integer iMicrosecondsinaday

  ldiLocale =@ SBLlocaledateinfo.new()
  iMicrosecondsinaday = 60 * 60 * 24 * 1000000

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = DATESTR(date.new(dt/iMicrosecondsinaday), "mmmm dd, yyyy", ldiLocale)
+ "  " + \
      TIMESTR(time.new(dt mod iMicrosecondsinaday), "hh:mm:ss.s am")

  cgi.output(sHTML_HEADER, 1)
  cgi.output("<html><body>The current date and time are:" + s +
  "</body></html>{d}{a}", 1)

end function ""
```

Preparing the Web Server to Run SIMPOL Programs

We recommend people use an Apache web server, if not for deployment at least for development. We are going to be using a solution stack called AMPPS which contains Apache, MySQL, PHP and various other services. There are other ways of installing Apache but these are outside the scope of this basic introduction.

Getting and Installing AMPPS

AMPPS is freely available to download, the latest version is available from https://www.ampps.com/download. Download and install AMPPS. Once you launch AMPPS it will look something like this:



C:/Program Files (x86)/Ampps
Apache started
MySql Started

Using a Web Server Other Than Apache

If you already have a web server running on your desktop, then you need to consult the documentation for it to find out how to run CGI programs using it. If the server runs as a service, you will need to open the Services applet from the Control Panel and then modify the service entry for your web server to allow it to Allow service to interact with desktop. This is needed if you wish to be able to debug your CGI programs in the Superbase NG IDE. Then set it up to execute programs that end in the file extension `.smp` as CGI programs. This may well need to know the location of the program used to run the applications and its name. The name of the program varies, depending on whether you are debugging or not. See the discussion of the Apache configuration below to learn about the various issues.

Configuring AMPSS Apache

Before we can run our program we also need to configure Apache. This section will discuss the changes that need to be made to the `httpd.conf` file.

Note
This section describes the minimal configuration required to develop and deploy web applications using SIMPOL and should not be considered to be a replacement for reading and understanding the documentation of the web server! Deploying a web server can be a complex operation depending upon the level of use it is expected to sustain. There is an

entire branch of the industry that handles the deployment and maintenance of high-availability web servers for e-commerce sites. Please don't confuse basic configuration of a single web server with the knowledge required to deploy a web server that should be handling thousands of hits per second on a continuous basis.

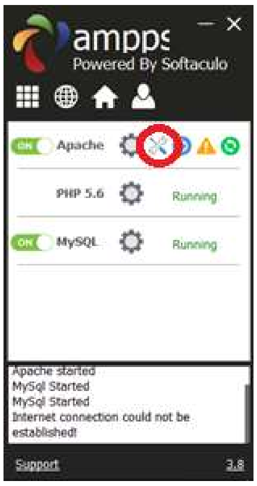
After AMPPS has been successfully installed, we need to find the AMPPS directory, this is normally found in: `C:\Program Files (x86)\Ampps`. In this directory there are a number of folders and sub-folders that will interest us:

- `Ampps > www > cgi-bin`
- `Ampps > Apache > logs`
- `Ampps > Apache > conf`

The `cgi-bin` directory is where our programs will normally reside. The `logs` directory is the place where the access and error logs are stored. The error log will be very important when trying to figure out why some program isn't working correctly. Often some useful information will show up in the log, such as an error message from SIMPOL. Finally, and what concerns us currently the `conf` directory is where the configuration files can be found.

To access the config files necessary simply right click on the `httpd.conf` file and open in a text editor.

However, it is also possible to access the necessary configuration file for Apache from the AMPPS client. To do this you go to the settings for the Apache web server (left click on the gear) and then press the configuration button (as highlighted below)



```
Apache started
MySql Started
MySql Started
Internet connection could not be
established!
```

Figure 31 Config button circled

At this point you will have opened the `httpd.conf` file and are ready to configure the server

The first item to change is the *`ServerAdmin`* parameter:

```
# ServerAdmin: Your address, where problems with the server should
# be e-mailed. This address appears on some server-generated pages,
# such as error documents. e.g. admin@your-domain.com
#
ServerAdmin johndoe@johndoe_world.com
```

Set the *`ServerAdmin`* parameter to your email address (or to whomever should be handling server problems).

The next parameter of importance is the *`ServerName`* parameter. For now, we will just change this to `localhost`. This can and should be changed to something else if you wish to deploy the server for more than development.

```
# ServerName gives the name and port that the server uses to
# identify itself. This can often be determined automatically, but
# we recommend you specify it explicitly to prevent problems during
# startup.
#
# If this is not set to valid DNS name for your host, server-
# generated redirections will not work.  See also the
# UseCanonicalName directive.
#
# If your host doesn't have a registered DNS name, enter its IP
# address here. You will have to access it by its address anyway,
# and this will make redirections work in a sensible way.
#
ServerName localhost
```

The *`DocumentRoot`* parameter should need no adjustment at this stage. You may wish to experiment with it later.

The *`ScriptAlias`* parameter for the `/cgi-bin/` directory is one of the more important parameters for our project. That and the associated *`Directory`* parameter should be set up to look like the following:

```
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the target directory are treated as applications and
# run by the server when requested rather than as documents sent to the
# client.  The same rules about trailing "/" apply to ScriptAlias
# directives as to Alias.
#
ScriptAlias /cgi-bin/ "{$path}/www/cgi-bin/"


# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory "C:/Program Files (x86)/Ampps/apache/htdocs">
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Restarting the Apache Web Server

Once the configuration is complete you have to restart the Apache Server, this is done by again pressing the settings cog in Ampps and then press the restart button

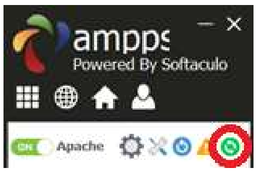
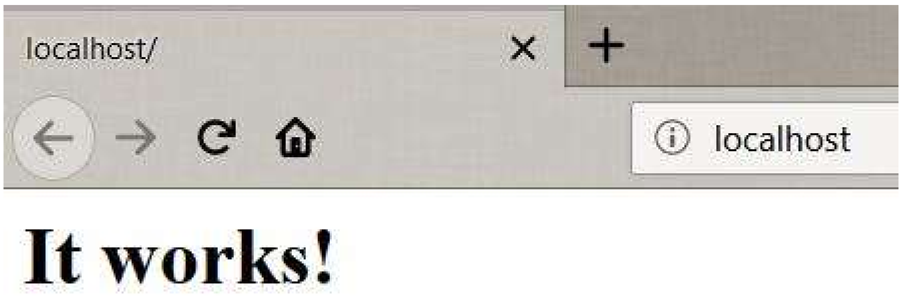


Figure 32 Restart server button

At this point you should be ready to start writing, running, and debugging SIMPOL web server programs. Before you go any further however, you should open your browser and enter the URL `http://localhost/`. If you don't see a screen like the one below, you may have made an error in your configuration of the Apache web server. You will need to correct that before you can go on. Check the Apache documentation, and look at the log files. They may tell you what is wrong.



It works!

Figure 33 Test screen for Apache

Debugging and Running Your Program

Finally, we are just about ready to actually run our web server program. There is only a little more preparation left to do so that we can run our program. First, to make our lives simpler, we need to add a secondary target for the build process. We do that by opening the Project Settings window and this time selecting the third tab labelled Targets, as shown in the picture below:

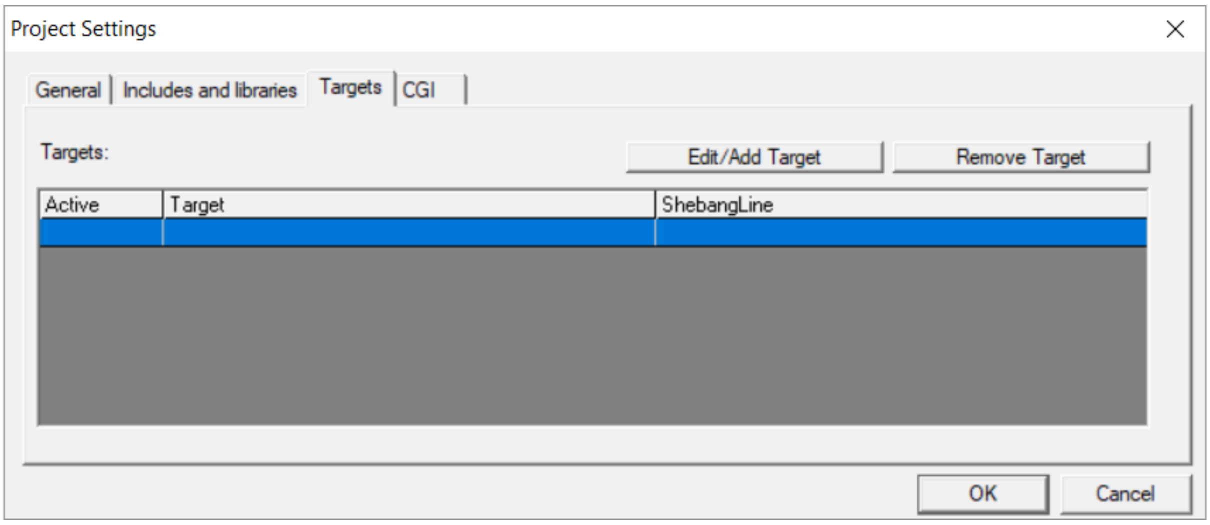


Figure 34 Project Settings Targets tab

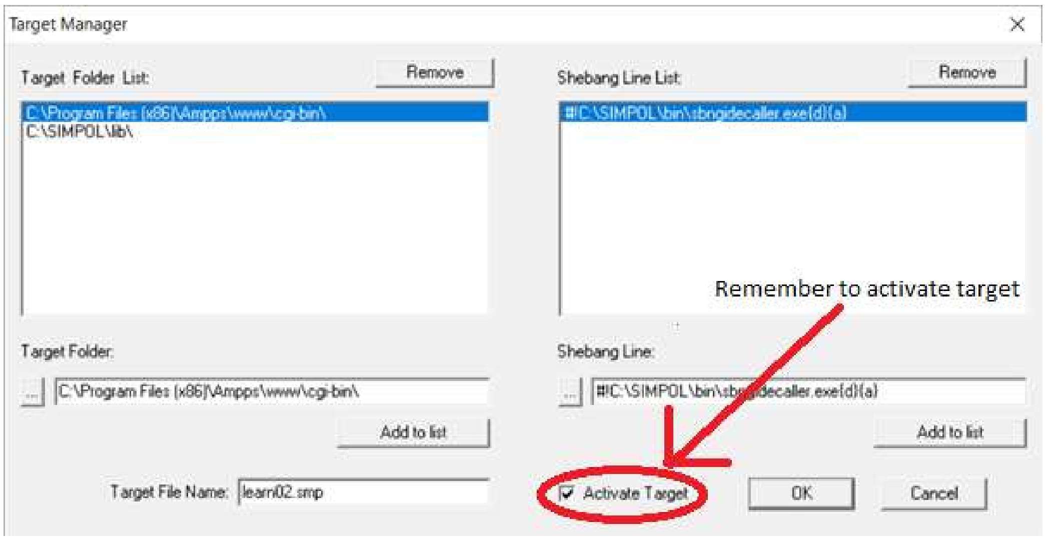
Click on the Edit/Add Target button. This displays the Target Manager window. On the left side of the window is a place to store common target directory names. On the right is a place to store common shebang lines.

Now you need to find the cgi-bin in your apache setup (this is your target folder for both the debugging and actual running of the software). For AMPPS this is normally found under:
`C:\Program Files (x86)\Ampps\www\cgi-bin\`

What's a shebang line?

In some operating systems, most notably UNIX, Linux, and now Mac OS-X, it is common to place a special type of comment at the beginning of a script that is marked as executable by the operating system. This comment must be the first line of the script and begins with a comment symbol, the hash mark (#) followed by the exclamation point symbol (!) (sometimes called the bang symbol – presumably from its use in comic books). This combination the "hash bang" has come to be known as the shebang. Directly following this character combination is the path and file name of the program that should be used to execute the script that follows. Programs that are aware of this convention and which support it can use this method of determining the correct processing program so that the script name alone is sufficient to run the script. The line *must* end with an end of line character that is correct for the target platform. On Windows and DOS machines this is the carriage return linefeed combination 0x0D0x0A. On UNIX, Linux, and Mac OS-X this is 0x0A alone.

Create a target such as the one shown in the picture below. You may wish to add the target directory and shebang lines to the lists since you will probably use them often. In this case we are creating a debug target. The program called `sbngidecaller.exe` is used to make direct callbacks into the IDE.


Target Manager

Target Folder List:
```
C:\Program Files (x86)\Ampps\www\cgi-bin\
C:\SIMPOL\lib\
```
Shebang Line List:
```
#!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}
```
Target Folder: C:\Program Files (x86)\Ampps\www\cgi-bin\

Shebang Line: #!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}

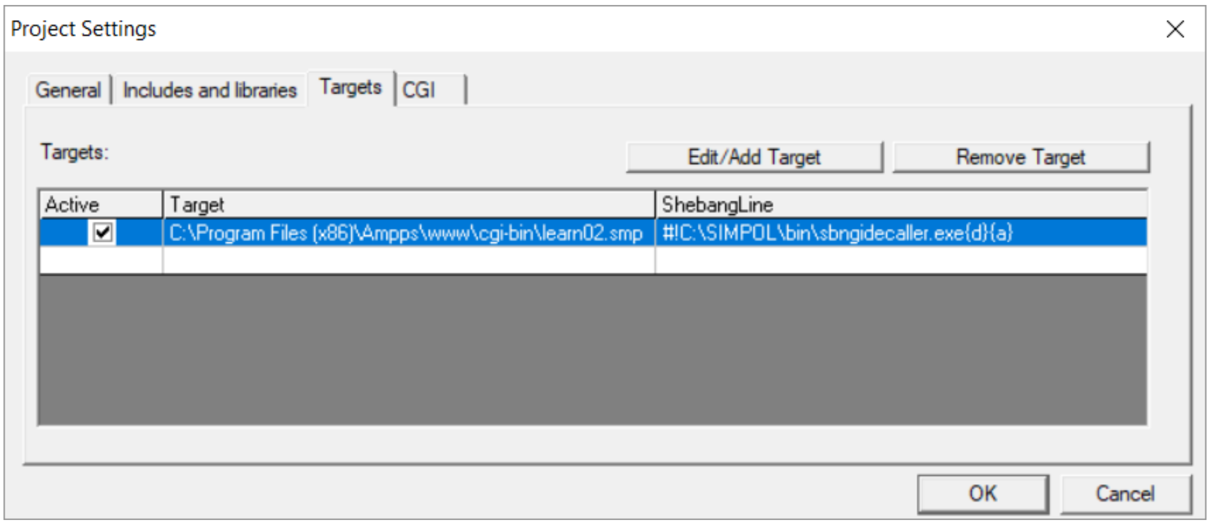
Target File Name: learn02.smp

[✓ Activate Target]

Figure 35 Creating our debug target for learn02.smp

The shebang line is: `#!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}`

The Target file name should be the same as the program, even if just to avoid confusion. Don't forget to activate the target! Then click the OK button to create the target and then the OK button to save the changes. Finally, press **Ctrl+B** to rebuild the project and create the secondary target.


Project Settings

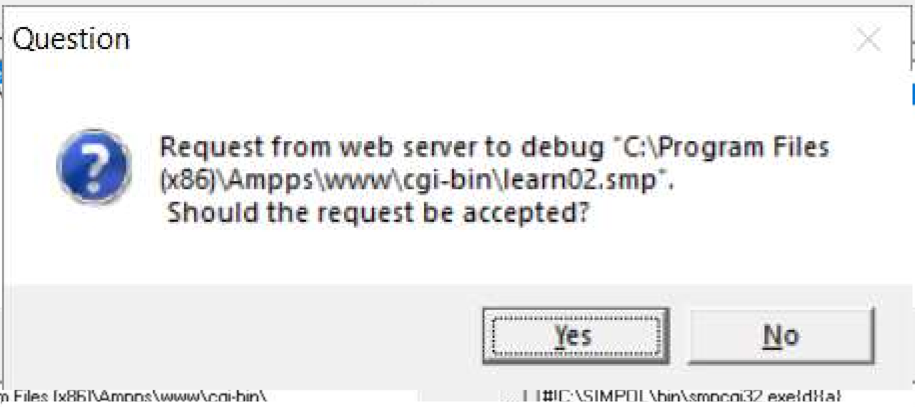
Targets:

Active	Target	ShebangLine
✓	C:\Program Files (x86)\Ampps\www\cgi-bin\learn02.smp	#!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}

Figure 36 Project Settings showing the debug target active

Now open a browser window and go to the following web address:
http://localhost/cgi-bin/learn02.smp

The following box should then pop-up, press yes.


Question

Request from web server to debug "C:\Program Files (x86)\Ampps\www\cgi-bin\learn02.smp". Should the request be accepted?

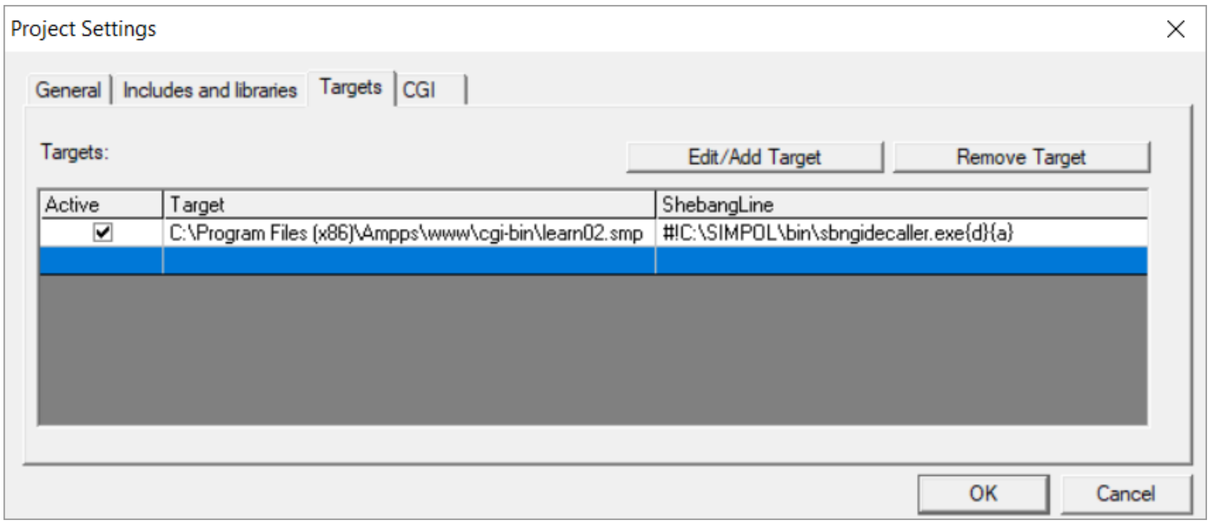
Figure 37 Request to debug program

This message is from the IDE indicating that it has received a request to debug a program. If that program is not the current one in your IDE the current project will be closed and the project associated with the program to be debugged will be opened. At this point everything runs exactly the same as when debugging normally.

Simply press **F5** to let the program just run through (this is where any bugs should turn up)

Once the debug has finished it will take you back to the browser, displaying an Internal Server Error. This is normal as we have only given it a debug not executed it properly

Now to display it in the webpage we need to add a second target, to do this again go to the project settings and click on the blank space

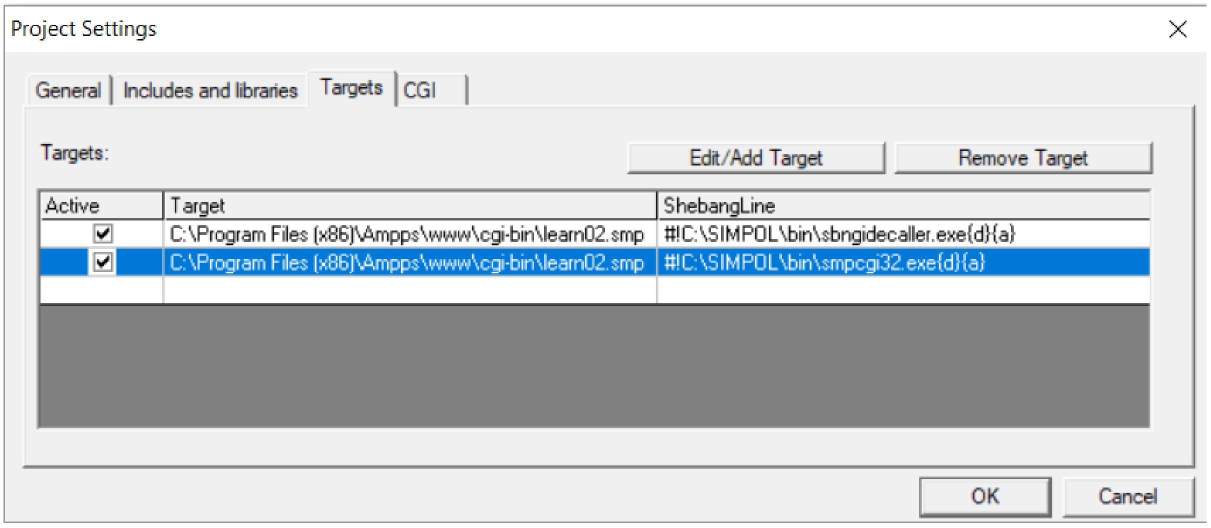

Project Settings

Active	Target	ShebangLine
✓	C:\Program Files (x86)\Ampps\www\cgi-bin\learn02.smp	#!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}

Figure 38 Project Settings Target for creating Second Target

Then press "Edit/Add Target" click on the target address we used previously but add the following shebang line: `#!C:\SIMPOL\bin\smpcgi32.exe{d}{a}`

Again remember to tick Activate Target and then click the OK button to create the target and then again to save the changes. The project settings should now look like this. Now all you need to do is to rebuild the project.

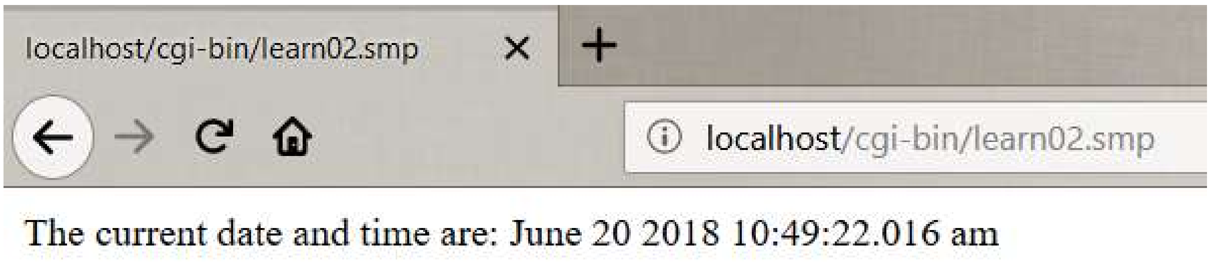


Active	Target	ShebangLine
✓	C:\Program Files (x86)\Ampps\www\cgi-bin\learn02.smp	#!C:\SIMPOL\bin\sbngidecaller.exe{d}{a}
✓	C:\Program Files (x86)\Ampps\www\cgi-bin\learn02.smp	#!C:\SIMPOL\bin\smpcgi32.exe{d}{a}

Figure 39 Target Manager settings for deployment target

Note
If more than one copy of the Superbase NG IDE are running concurrently then the first window that pops up will not be from the IDE about a request to debug, it will be from the `sbngidecaller.exe` program asking which copy of the IDE should receive the request to debug. Select the appropriate entry and then the debugging request message will appear.

Now you can go back to the browser and refresh the page. It should now execute the program and display the time and date (the exact time and date will of course be different for you)



```
The current date and time are: June 20 2018 10:49:22.016 am
```

Figure 40 learn02 successful execution

Optionally, you can now deactivate the debug target as it is no longer necessary

Summary

In this part we have learned how to:

- Save an existing project as a new project
- Add an external source file to our program with the `include` statement
- Add include directories to our project definition

- Use a `constant` in our code
- Convert a program to work as a web server application
- Retrieve, install, and minimally configure the Apache web server
- Work with targets in the IDE
- Debug a web server application

In the next part we will learn about debugging into the source code projects provided for most of the SIMPOL libraries.

Debugging Into Library Source Code

In the first part, we built our first basic program in SIMPOL and learned how to use the IDE to do various tasks, including basic debugging. In this part, we will learn how to debug into the source code of one of the supplied libraries.

Debugging Revisited

Debugging a program is an important aspect of the development process. Since so many parts of Superbase NG are delivered as libraries, it is important to be able to assess what happens to your program when it goes into one of those libraries, especially if you think that there is a flaw in the library itself. Another important aspect is to learn how the libraries are written, and see how they work. This can be a useful tool for learning about the programming language. Using the Superbase NG IDE you can debug into any library for which the source code (as a project) is available.

Adding Library Source Code

We are going to be using the project learn01, so we should open this. First close whatever your current project is (File>Close Project). Now open learn01.smp (File>Open Project)

To add source code from another project to the current one, in the project tree view right-click the root node and select “Import Modules From Project”

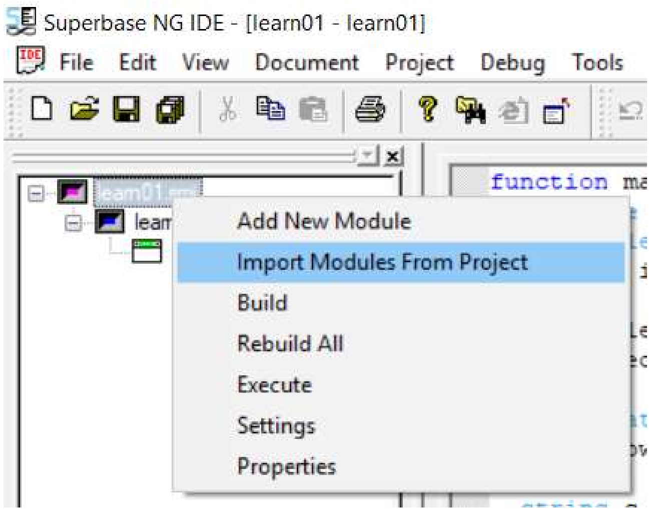
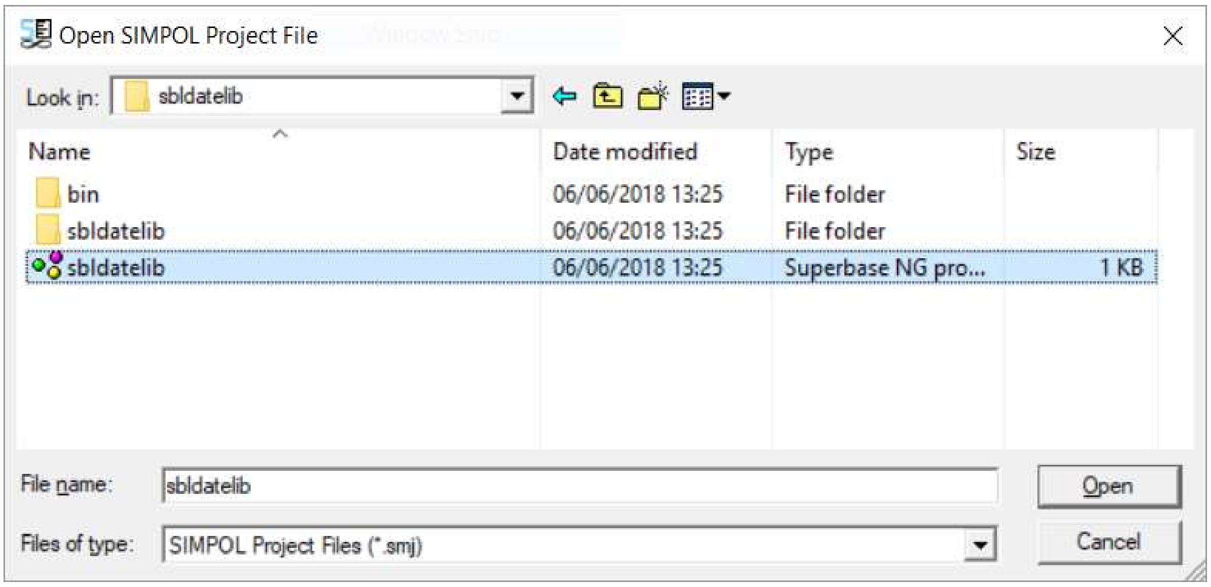


Figure 41 Project context menu

In the resulting window find: `C:\SIMPOL\projects\libs\sbldatelib`



Open SIMPOL Project File

Look in: sbldatelib

Name	Date modified	Type	Size
bin	06/06/2018 13:25	File folder	
sbldatelib	06/06/2018 13:25	File folder	
sbldatelib	06/06/2018 13:25	Superbase NG pro...	1 KB

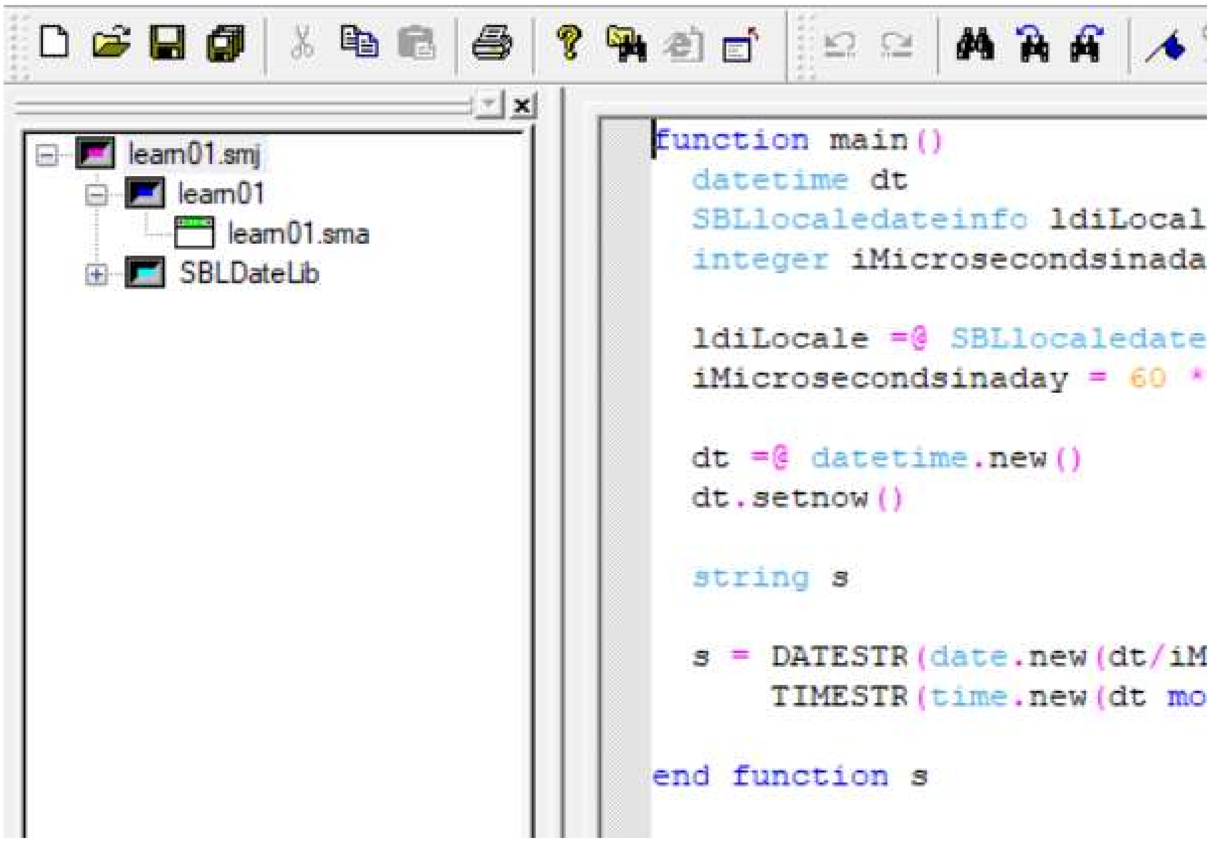
File name: sbldatelib

Files of type: SIMPOL Project Files (*.smj)

Figure 42 Open Project window

And then open the `SBLDateLib.smj`

This will add the library project into the current project as a module. Projects loaded in this way cannot be modified, the source code is read-only. Once it has been added, the IDE will retrieve the function and type information, and add it to the tree



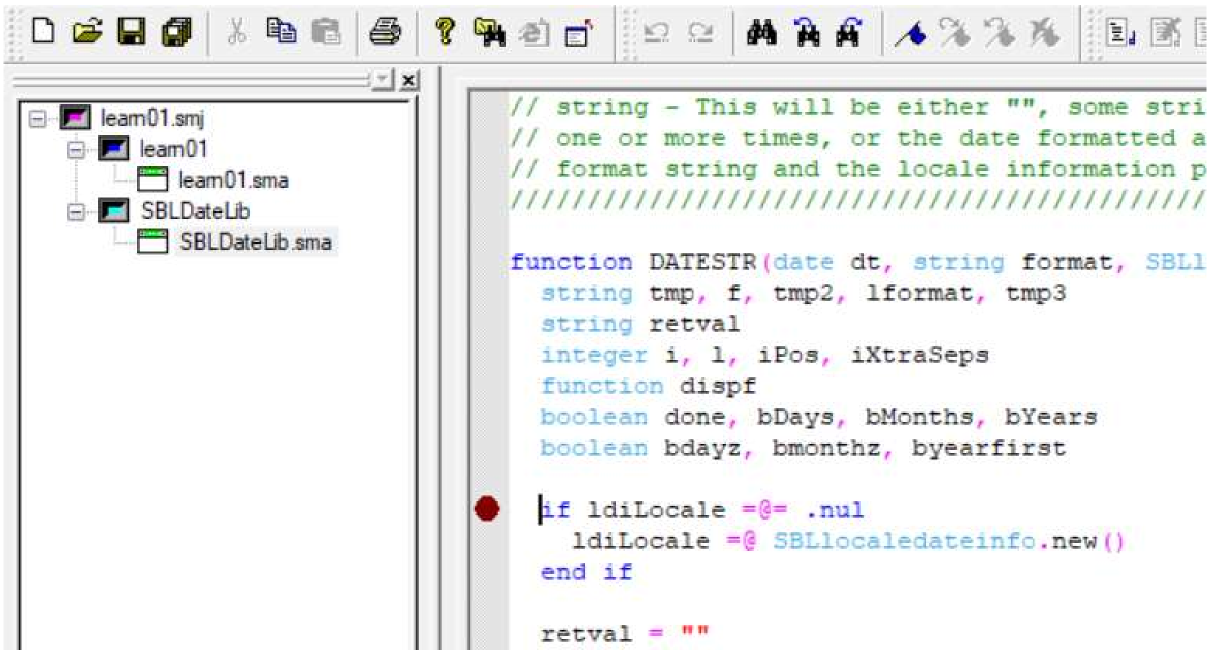
```
learn01.smj
  learn01
    learn01.sma
  SBLDateLib
```

```
function main()
  datetime dt
  SBLlocaledateinfo ldiLocal
  integer iMicrosecondsinada

  ldiLocale =@ SBLlocaledate
  iMicrosecondsinaday = 60 *

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = DATESTR(date.new(dt/iM
      TIMESTR(time.new(dt mo

end function s
```

Figure 43 Project after importing a module for debugging

Now that the project code has been imported, we can now open the source file in the editor. First expand the entry for `SBLDatetLib` and then double click on `SBLDateLib.sma`.

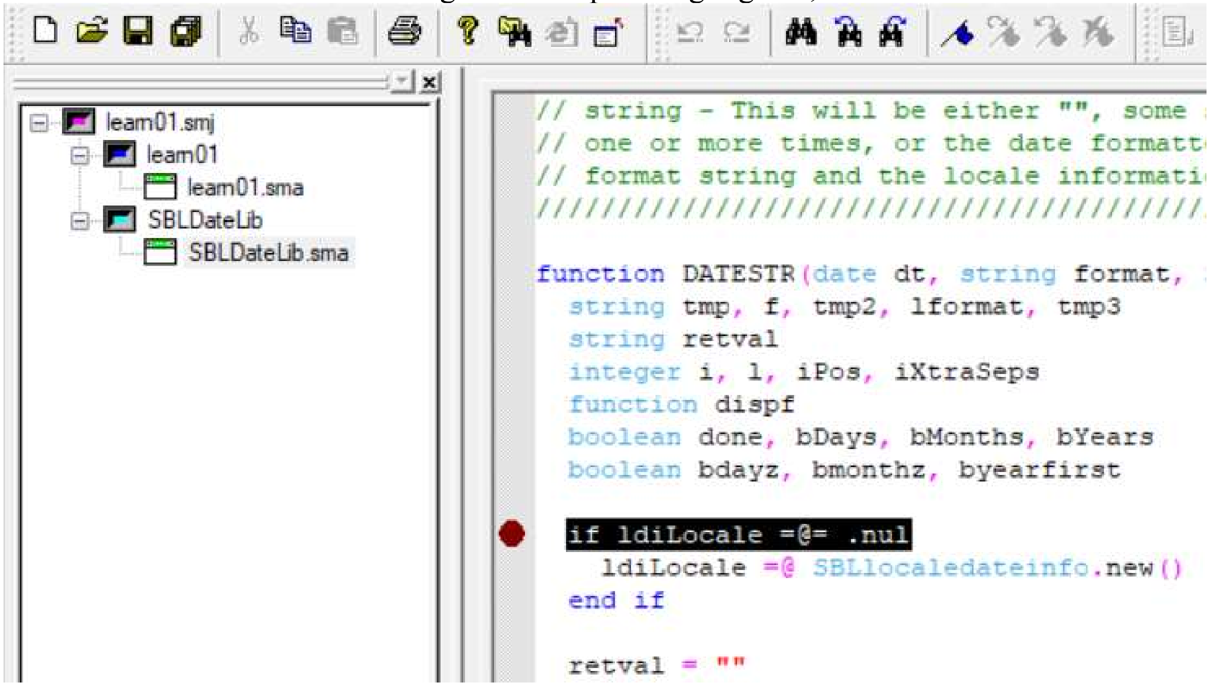
To debug inside the call to the library function we will place a break point at the beginning of the function where indicated below. This is done by clicking on the line and pressing F9 (or Debug>Insert/Remove Breakpoint).



```
// string - This will be either "", some stri
// one or more times, or the date formatted a
// format string and the locale information p
///////////////////////////////////////////

function DATESTR(date dt, string format, SBLl
  string tmp, f, tmp2, lformat, tmp3
  string retval
  integer i, l, iPos, iXtraSeps
  function dispf
  boolean done, bDays, bMonths, bYears
  boolean bdayz, bmonthz, byearfirst

  if ldiLocale =@= .nul
    ldiLocale =@ SBLlocaledateinfo.new()
  end if

  retval = ""
```

Figure 44 Setting a break point in a library function

To enter debugging press **F4**. Then press **F5** to run through the program. Program execution should halt with the line containing the break point highlighted, as shown below:



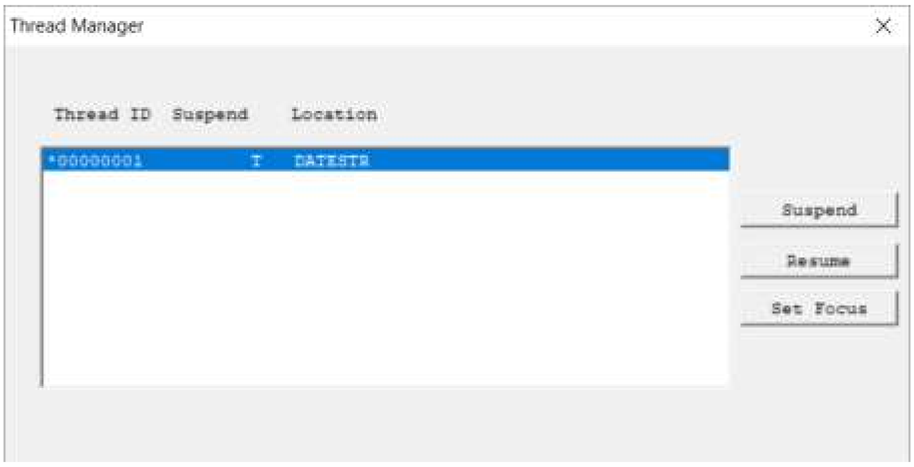
```
// string - This will be either "", some
// one or more times, or the date formatt
// format string and the locale informati
//////////////////////////////////////////

function DATESTR(date dt, string format,
  string tmp, f, tmp2, lformat, tmp3
  string retval
  integer i, l, iPos, iXtraSeps
  function dispf
  boolean done, bDays, bMonths, bYears
  boolean bdayz, bmonthz, byearfirst

  if ldiLocale =@= .nul
    ldiLocale =@ SBLlocaledateinfo.new()
  end if

  retval = ""
```

Figure 45 Code execution halted at the break point

At this point, we can start single-stepping through the code (using **F11**), examine the variables, and evaluate what is happening to the code. In the beginning of the function most of the variables will be equal to `.nul`, since they have been declared but nothing has yet been assigned to them.

Tip

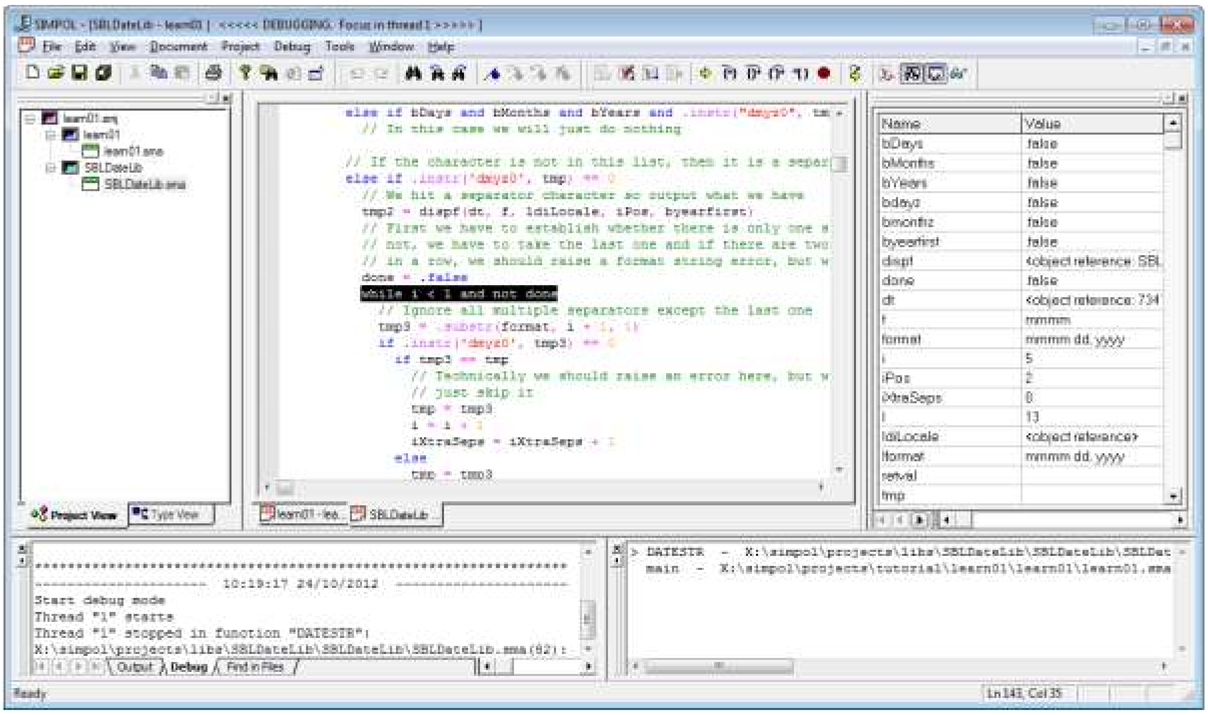
If you are debugging code that is working with a multi-threaded system, such as that provided by the SIMPOL Application Framework, then you may need to switch to the correct thread before any debugging commands will work or the variables are shown in the list. To do this select Debug → Thread Manager and in the window select the correct thread, then click on the Set Focus button.



```
Thread ID  Suspend   Location
*00000001        T   DATESTR
```

Figure 46 Thread Manager dialog window

Further into the function, as shown in the following image, the variables can be seen to the right in the watch window and the call stack is shown in the window below that.



```
    else if bDays and bMonths and bYears and .instr("dmyz0", tm
      // In this case we will just do nothing

    // If the character is not in this list, then it is a separ
    else if .instr("dmyz0", tmp) == 0
      // We hit a separator character so output what we have
      tmp2 = dispf(dt, f, ldiLocale, iPos, byearfirst)
      // First we have to establish whether there is only one s
      // not, we have to take the last one and if there are two
      // in a row, we should raise a format string error, but w
      done = .false
      while i < l and not done
        // Ignore all multiple separators except the last one
        tmp3 = .substr(format, i + 1, 1)
        if .instr("dmyz0", tmp3) == 0
          if tmp3 == tmp
            // Technically we should raise an error here, but w
            // just skip it
            tmp = tmp3
            i = i + 1
            iXtraSeps = iXtraSeps + 1
          else
            tmp = tmp3
```

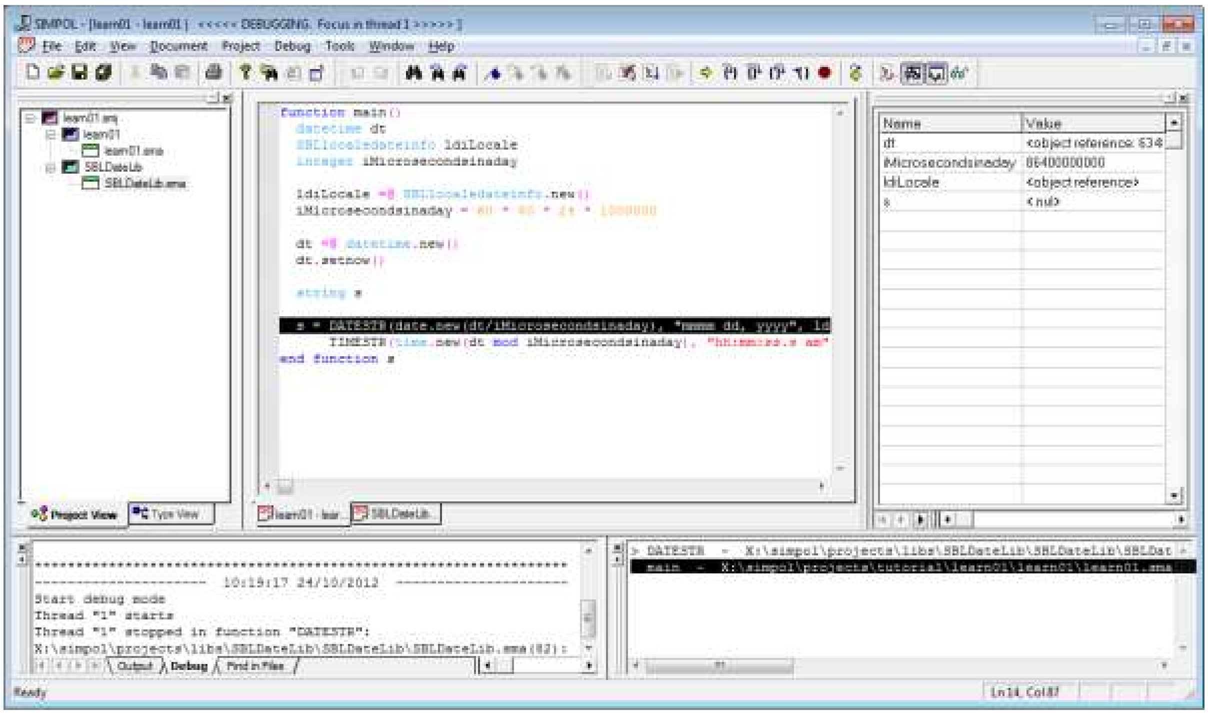
Name	Value
bDays	false
bMonths	false
bYears	false
bdayz	false
bmonthz	false
byearfirst	false
dispf	<object reference: SBL
done	false
dt	<object reference: 734
f	mmmm
format	mmmm dd, yyyy
i	5
iPos	2
iXtraSeps	0
l	13
ldiLocale	<object reference>
lformat	mmmm dd, yyyy
retval	
tmp	

```
**********************************************************************
---------------------  10:19:17 24/10/2012  ---------------------
Start debug mode
Thread "1" starts
Thread "1" stopped in function "DATESTR":
X:\simpol\projects\libs\SBLDateLib\SBLDateLib\SBLDateLib.sma(82):
```

```
> DATESTR  -  X:\simpol\projects\libs\SBLDateLib\SBLDateLib\SBLDat
  main  -  X:\simpol\projects\tutorial\learn01\learn01\learn01.sma
```

Figure 47 Debugging show the variables and call stack

By double-clicking on a function further down the call stack, the point in the code that called the next function further up the call stack can be displayed, and the state of the variables in that function can be shown.



```
function main()
  datetime dt
  SBLlocaledateinfo ldiLocale
  integer iMicrosecondsinaday

  ldiLocale =@ SBLlocaledateinfo.new()
  iMicrosecondsinaday = 60 * 60 * 24 * 1000000

  dt =@ datetime.new()
  dt.setnow()

  string s

  s = DATESTR(date.new(dt/iMicrosecondsinaday), "mmmm dd, yyyy", ld
      TIMESTR(time.new(dt mod iMicrosecondsinaday), "hh:mm:ss.s am"
end function s
```

Name	Value
dt	<object reference: 634
iMicrosecondsinaday	86400000000
ldiLocale	<object reference>
s	<null>

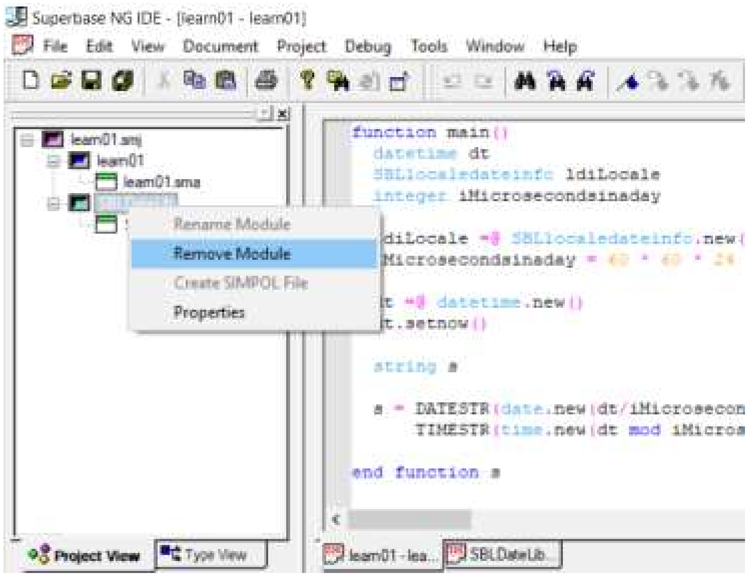
```
**********************************************************************
---------------------  10:19:17 24/10/2012  ---------------------
Start debug mode
Thread "1" starts
Thread "1" stopped in function "DATESTR":
X:\simpol\projects\libs\SBLDateLib\SBLDateLib\SBLDateLib.sma(82):
```

```
> DATESTR  -  X:\simpol\projects\libs\SBLDateLib\SBLDateLib\SBLDat
  main  -  X:\simpol\projects\tutorial\learn01\learn01\learn01.sma
```

Figure 48 Debugging showing the variables at a different call stack position

Removing Library Source Code

Once the debugging exercise with the library source code is complete, it is beneficial to remove the project from the tree, since each time debugging is entered, the IDE must analyze the source code from the library project as well as the development project, which slows down development. To remove the project, right-click on the module node in the project tree, and select the menu item Remove Module from the pop-up menu.



```
function main()
  datetime dt
  SBLlocaledateinfo ldiLocale
  integer iMicrosecondsinaday

 diLocale =@ SBLlocaledateinfo.new(
 Microsecondsinaday = 60 * 60 * 24

 t =@ datetime.new()
 t.setnow()

  string s

  s = DATESTR(date.new(dt/iMicrosecon
      TIMESTR(time.new(dt mod iMicros

end function s
```

Figure 49 Removing the imported source code module

Summary

In this part we have learned how to:

- Import a library project to allow debugging into its source code
- Use the Thread Manager to select the correct thread
- Remove a library project module once it is no longer required

Now its time to open up some of the sample projects and try them out. Have fun!