

Superbase NG IDE Quick Start Manual

How to Use the Superbase NG IDE

Neil Robinson

Superbase NG IDE Quick Start Manual: How to Use the Superbase NG IDE

by Neil Robinson

Copyright © 2001-2017 Superbase Software Limited

All rights reserved. The programs and documentation in this book are not guaranteed to be without defect, nor are they declared to be fit for any specific purpose other than instruction in the use of the programming language SIMPOL. It is entirely possible (though not probable) that use of any sample program code in this book could reformat your hard disk, disable your computer forever, fry your dog in a microwave oven, and even cause a computer virus to infect you by touching the keyboard, though none of these things is terribly likely (after all, almost *anything* is possible, it is just that most things are extremely improbable).

Table of Contents

1. Introduction	1
2. Getting Started with the Superbase NG IDE	3
Creating Our First Project	4
Writing Our First Program	6
Building and Testing	8
Making Incremental Improvements	11
Summary	15
3. Writing Web Server Programs With SIMPOL	17
Converting Our Previous Project	17
Preparing the Web Server	21
Using Another Web Server	21
Getting and Installing Apache	22
Configuring Apache	22
Configuring the Service	24
Restarting the Apache Web Server	26
Debugging and Running	27
Summary	30
4. Debugging Into Library Source Code	31
Debugging Revisited	31
Adding Library Source Code	31
Debugging Library Source Code	32
Removing Library Source Code	35
Summary	36

Chapter 1. Introduction

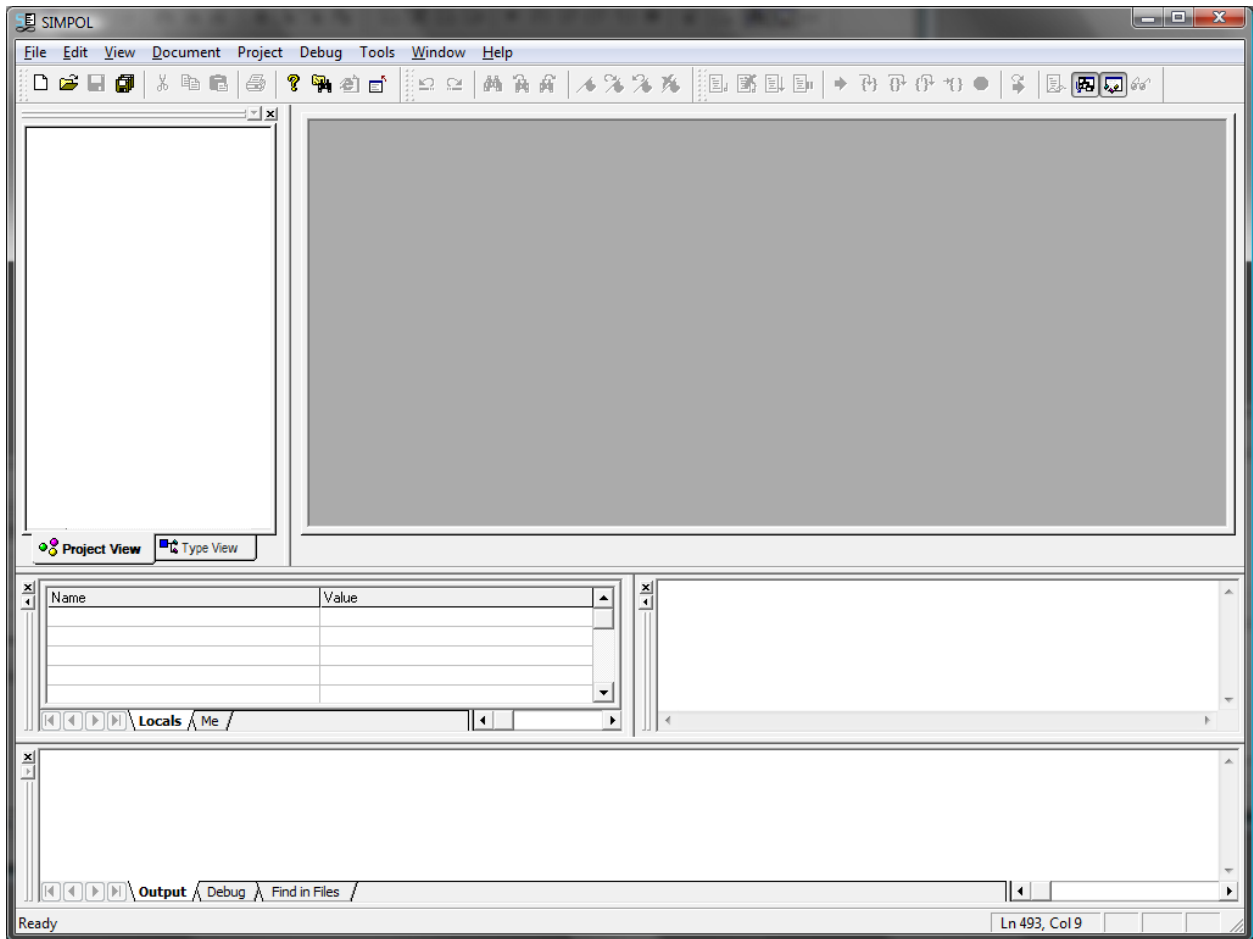
This book is intended to provide a quick start guide for using the new Superbase NG IDE. Although in the process some SIMPOL programming is covered, that is not the primary purpose of this book. Other books are provided that cover SIMPOL programming.

The chapters 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 the Apache web 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.

Chapter 2. 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:



The 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 (there are no tabs in this picture since there are no documents opened). Next 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 lower left corner 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

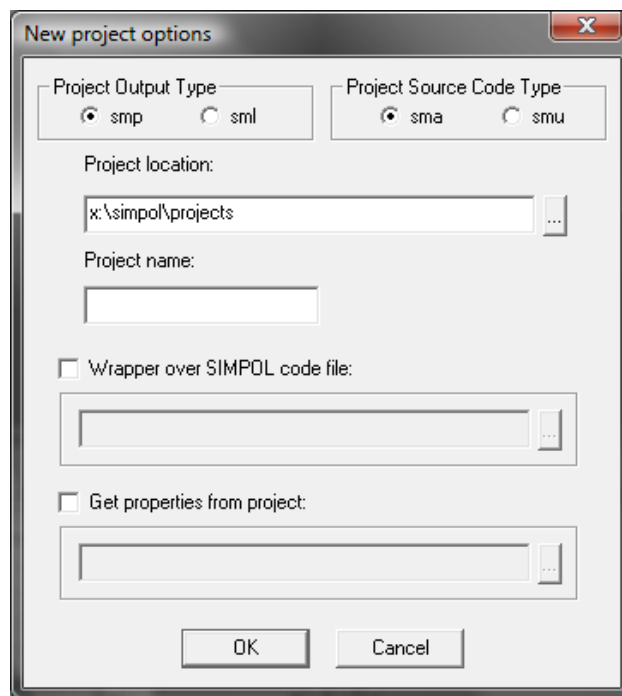
the Call Stack panel would show the three functions in the reverse order that we called them. Also, the Watch panel would contain the current state of the variables in the `init_databases()` function. If we then clicked on one of the earlier functions, the Watch window would then show us the state of 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 and arranged in a different manner that may better suit the user. Please feel free to try things out.

Now that we have had a look around, the smart thing to do would be to actually build something. In the next section, we will build our first project, a very basic program that is designed to teach us more about the development environment.

Creating Our First Project

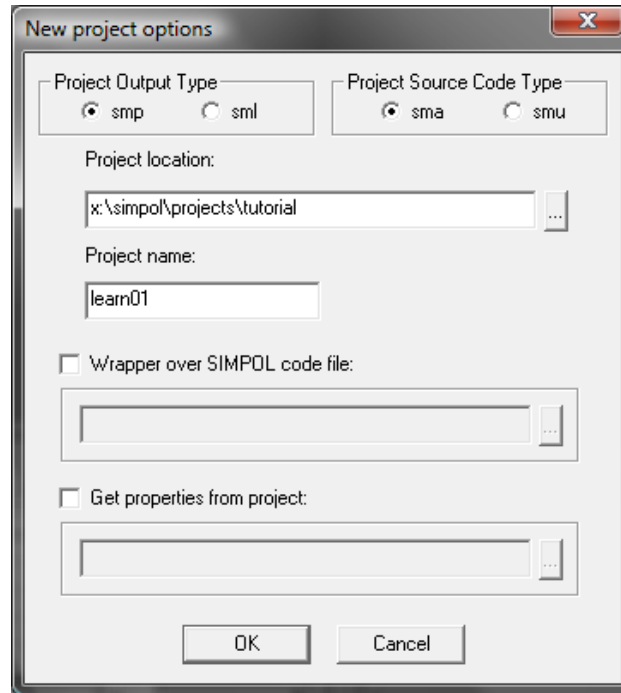
To create a new Superbase NG project, select the New Project from the File menu. At that point, the following window will be shown:



The New Project Options window

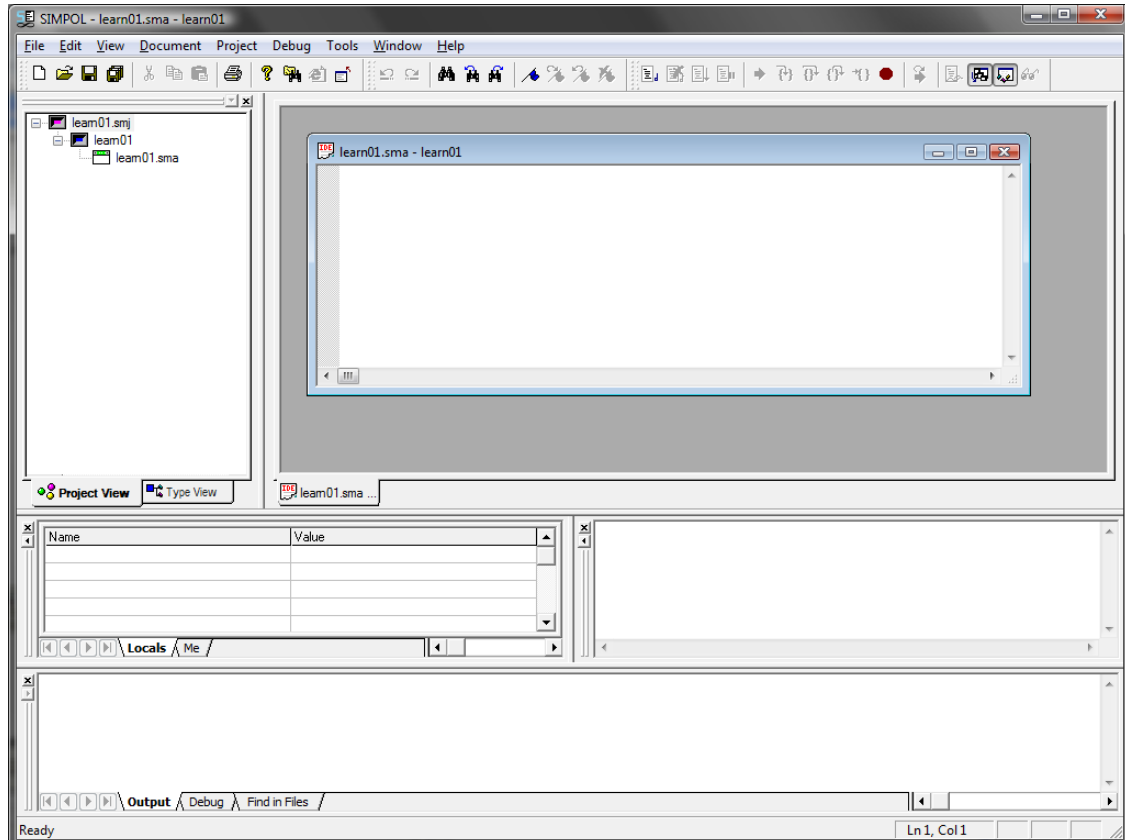
In this window we define the project name, project output type (sml or smp), source code file type (sma or smu), where the project should be located and what it should be called. The options in the lower half of the window are advanced options that we won't cover at this time. In this example we will 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:



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:



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 chose 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()

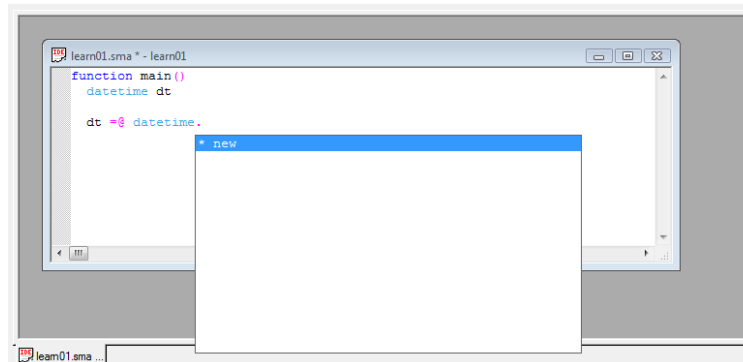
  string s

  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.

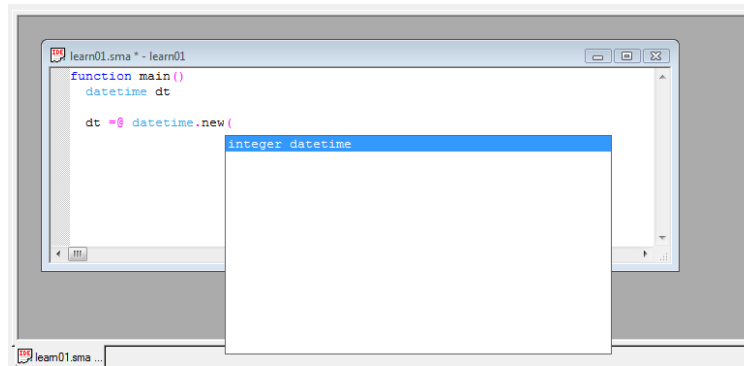
As you type in the first part of the code, as shown in the following picture, a number of things may occur to you. First, the various words and punctuation in the program appear in different colors. Color-coding of the source code is quite common today and assists the reader in immediately being able 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.



The first portion of the learn01 program

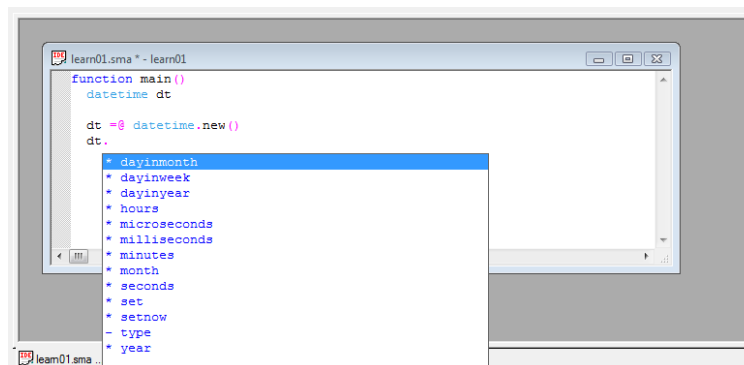
The picture above shows the inline programming help for the datetime type. Since the new() method is the only item in the list, it is already pre-selected. Merely by pressing the **tab** key the text will be entered at the cursor position. Regular use of this feature can greatly reduce the time it takes to write programs, as well as reducing the number of typing errors.

Once the new method name has been entered when the open parentheses is typed the inline help shows the arguments for the call to the method, as can be seen from the following picture.



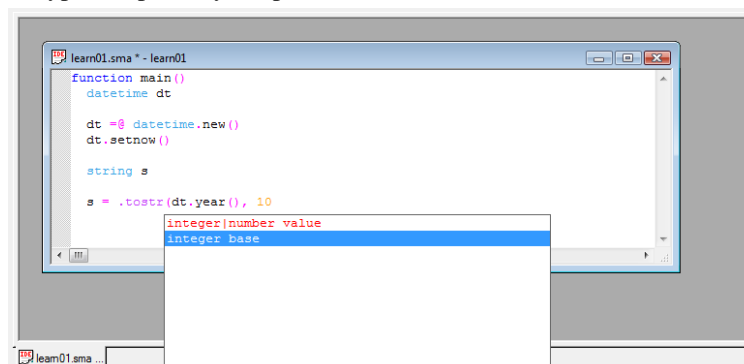
The inline help for the new() method

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.



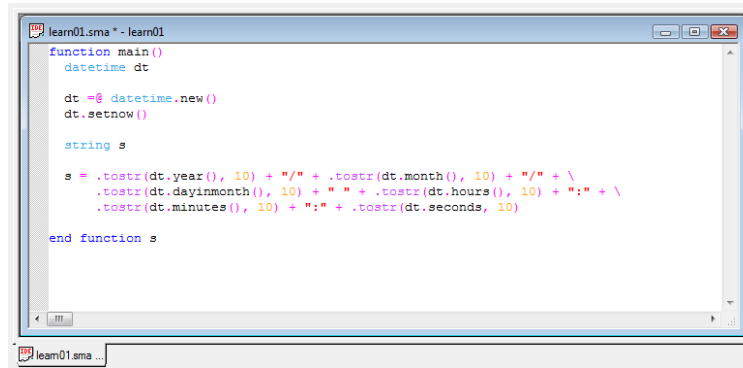
The inline help for the datetime object

Every component in SIMPOL has inline help, such as the intrinsic function `.tostr()` as seen in this picture. 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.



The inline help for the .tostr() intrinsic function

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)

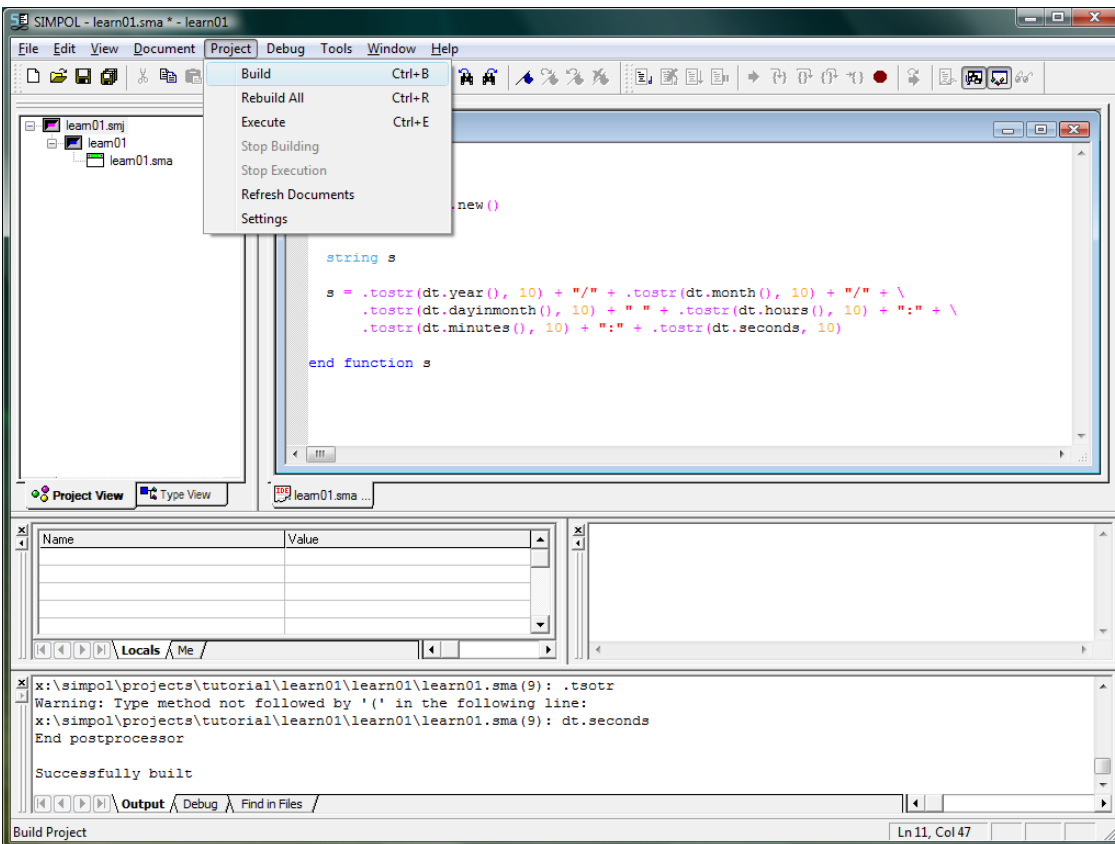
end function s

```

The complete source code for the first project

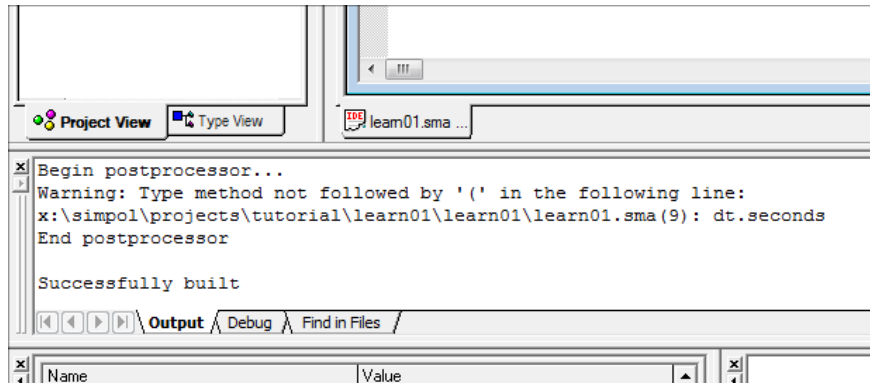
Building and Testing Our First Program

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.



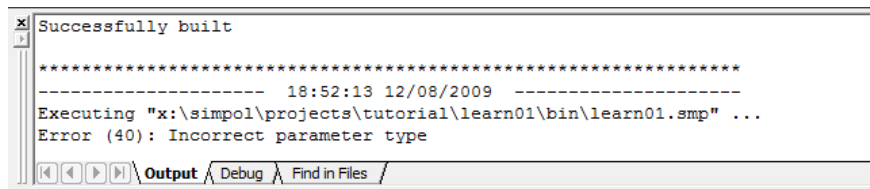
Building the first project

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



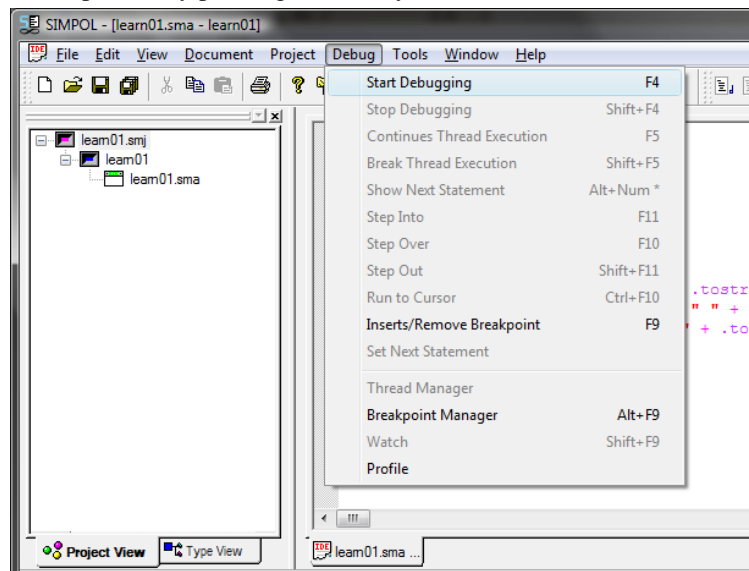
The first project successfully compiled

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.



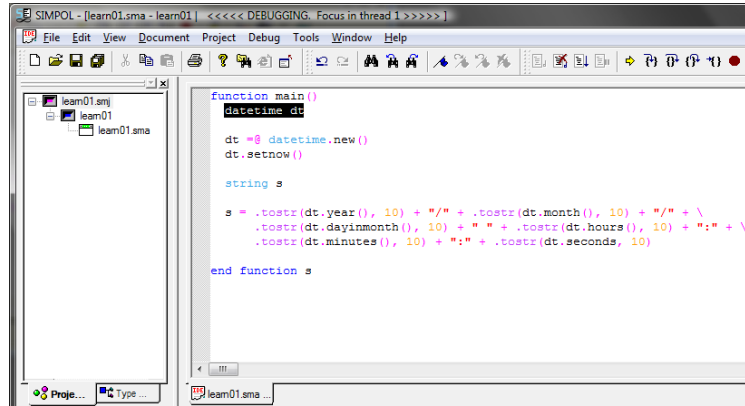
The first project fails with an error

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.



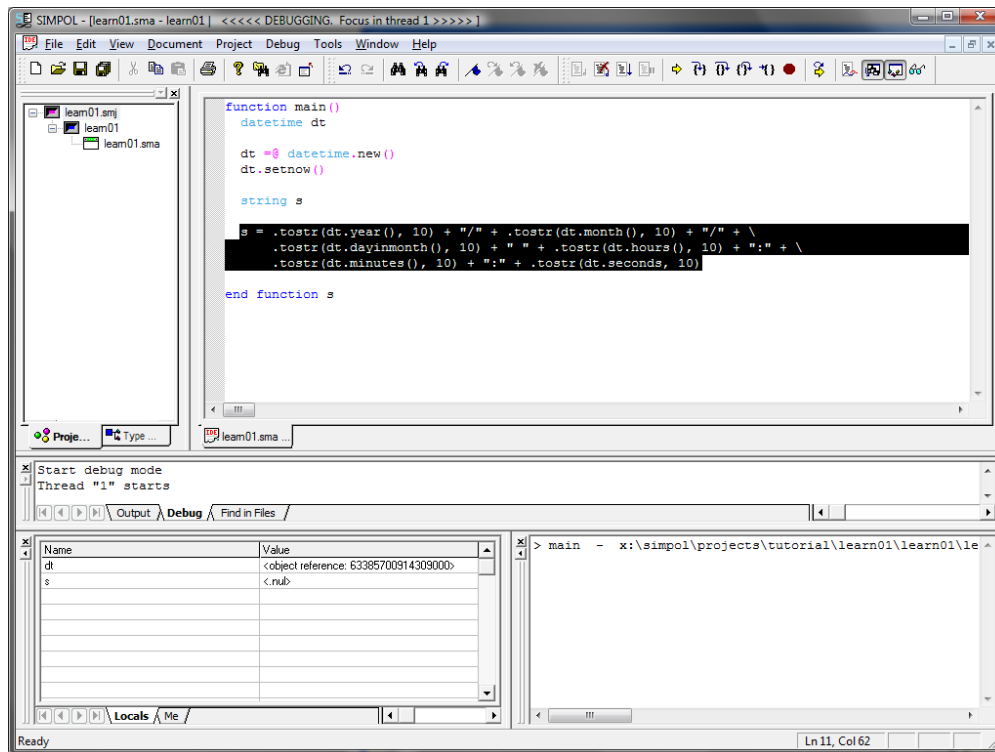
Starting the debugger

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:



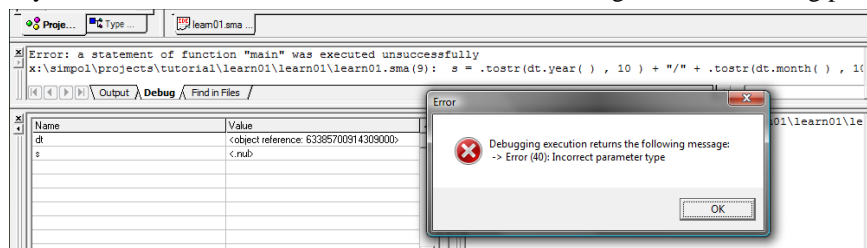
The debugger stopped on the first statement

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



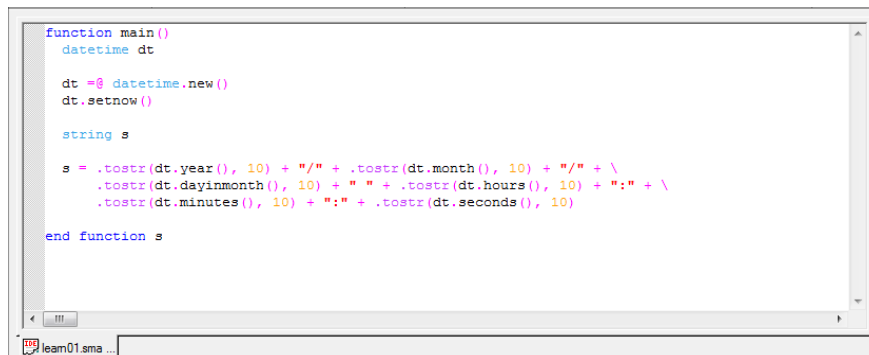
The debugger stopped on the last statement inside the function

Pressing the **F10** key once more will result in an error that looks something like the following picture:



The program is halted with an error

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.



```
function main()
  datetime dt

  dt :=& datetime.new()
  dt.setnow()

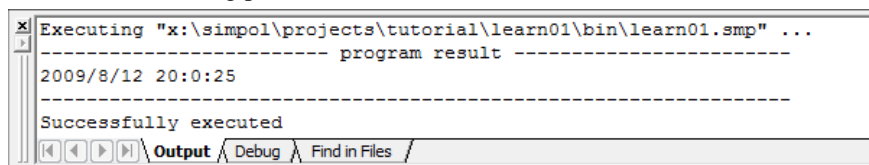
  string s

  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
```

The corrected program source

Now by pressing **Ctrl+B** (or by selecting the appropriate item from the menu) we can rebuild the program and by then pressing **Ctrl+E** (or a valid alternative) we can execute the program. This time the program runs successfully without errors, as we can see from the resulting picture.



```
Executing "x:\simpol\projects\tutorial\learn01\bin\learn01.smp" ...
----- program result -----
2009/8/12 20:0:25
-----
Successfully executed
```

The program has run successfully

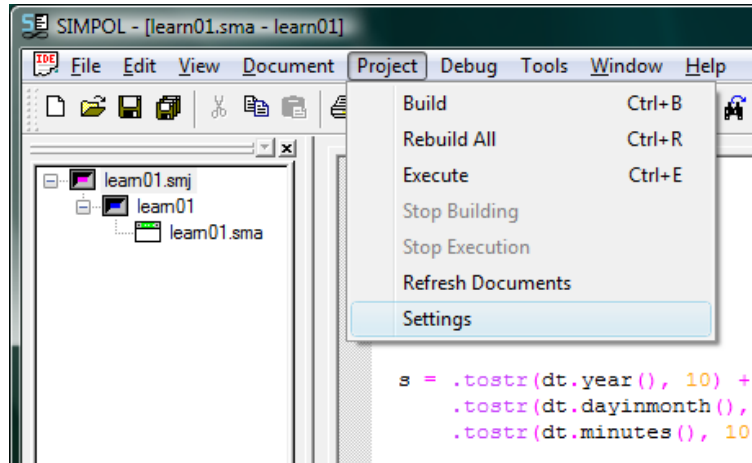
Upon careful examination of the output from the program, however, we can already see that there is still some improvement that can be made over the current version. The program output the string `2009/8/12 20:0:25`. Although the date might be considered acceptable, the time is certainly not going to 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.

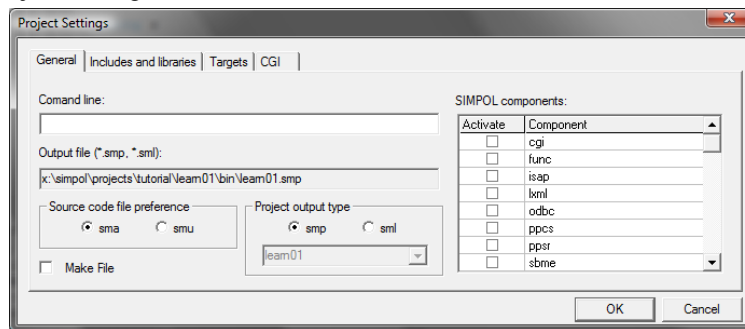
There is currently a large and ever-growing library of pre-designed functionality supplied with SIMPOL and in most cases the fully commented source code of the library is also provided. Pre-compiled libraries are located by default in the `lib` subdirectory of the place where Superbase NG was installed. 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.



Opening the Project Settings window via the menu

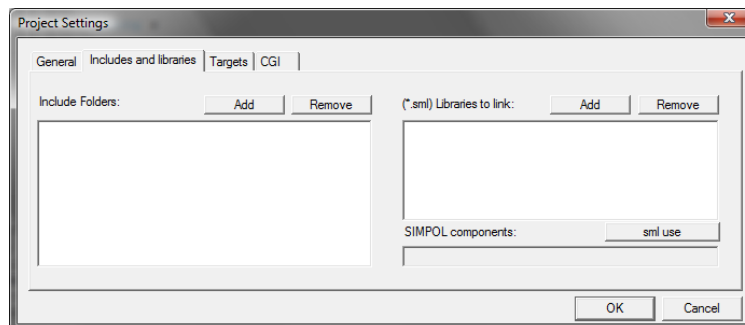
This will display the Project Settings window.



The Project Settings window

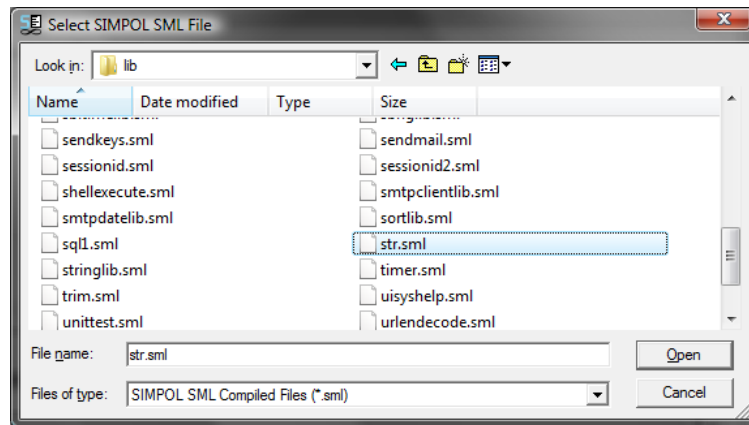
This window is extremely important for creating powerful and successful applications using SIMPOL. The initial tab allows the setting of the source code file preference, assigning of command line parameters when running and debugging in the IDE, and also provides a method of selecting the SIMPOL components 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.



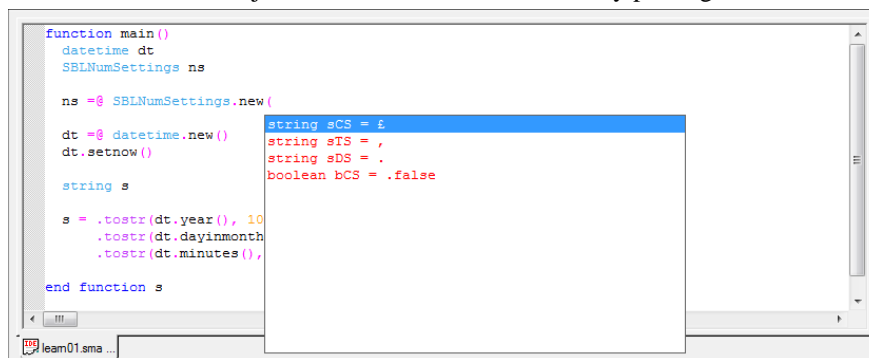
The Project Settings Includes and libraries tab

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`.



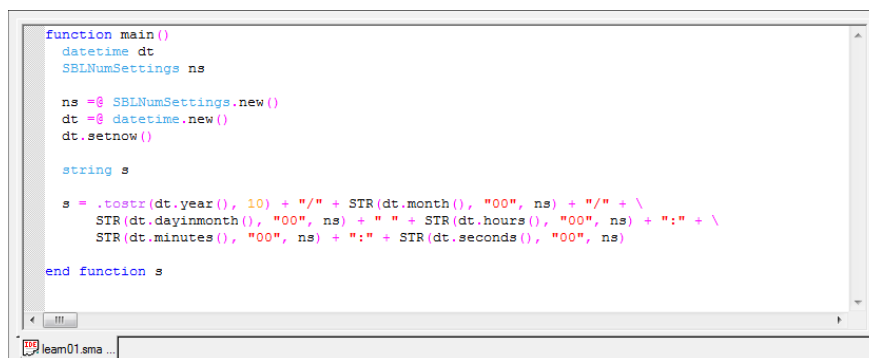
The file selection window for `STR.sml`

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.



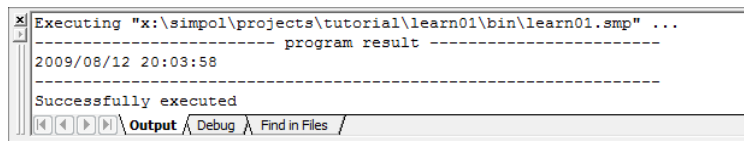
The inline help for the `SBLNumSettings` type

Continue modifying the code until it looks like the source code in the image that follows. We will replace nearly all instances of the function `.tostr()` with the function `STR()`. This will give greater flexibility when formatting our numbers.



The reworked source code using the `STR()` function

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).



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 format common in English-speaking countries (and possibly elsewhere). 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. Modify the source code to look that shown in the program listing that follows:

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

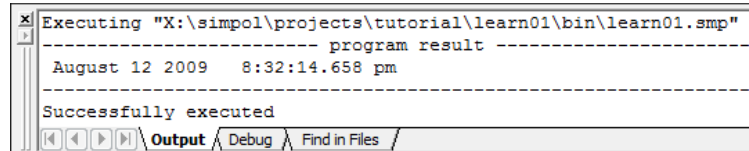
  ldiLocale =@ SBLlocaledateinfo.new()
  iMicrosecondsinaday = 60 * 60 * 24 * 1000000

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

  string s

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

Again, just as with the `STR()` function, the date functionality requires some formatting information that was globally present in SBL but which must be provided explicitly in SIMPOL. This time after building and executing the program we can see that we are now able to finely control the formatting of the output.



The screenshot shows a window titled "Executing 'X:\simpol\projects\tutorial\learn01\bin\learn01.smp' .". The output text is as follows:

```
-----  
program result -----  
August 12 2009   8:32:14.658 pm  
-----  
Successfully executed
```

At the bottom of the window, there are navigation buttons and tabs labeled "Output", "Debug", and "Find in Files".

The output of the new program using date and time functions

Summary

In this chapter 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)

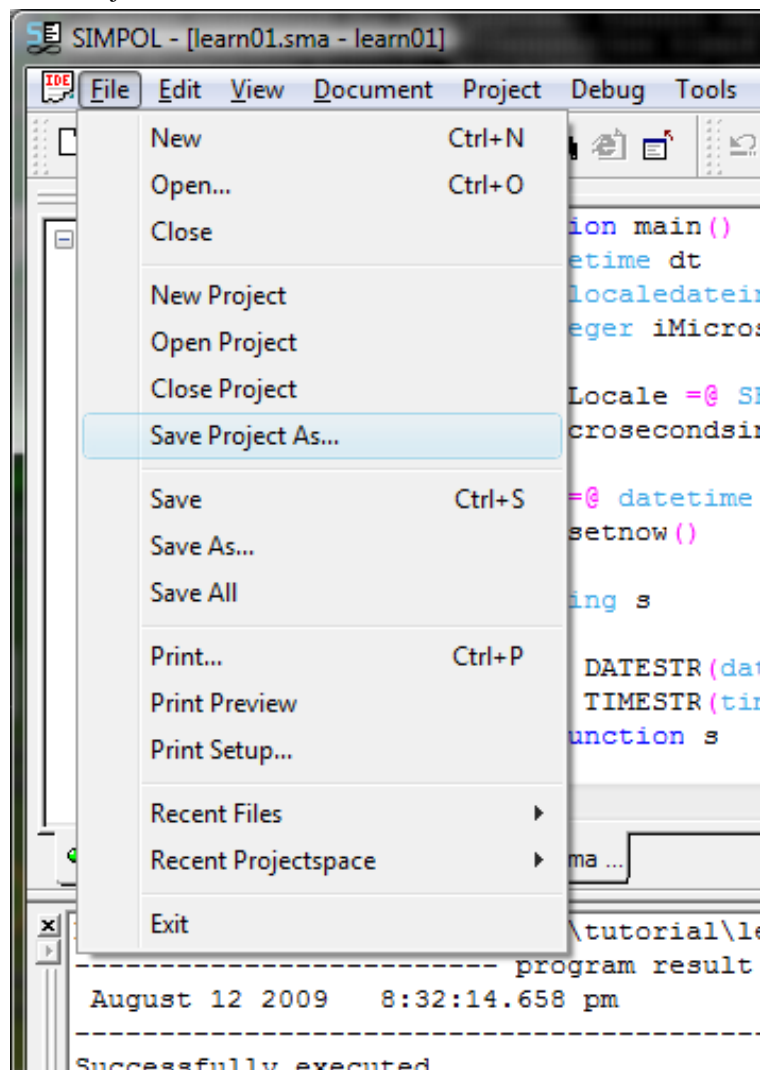
In the next chapter we will take our current project and learn how to modify it to output the results in a web page as a web server application.

Chapter 3. Writing Web Server Programs With SIMPOL

In the previous chapter, we built our first basic program in SIMPOL and learned how to use the IDE to do various tasks. In this chapter, 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

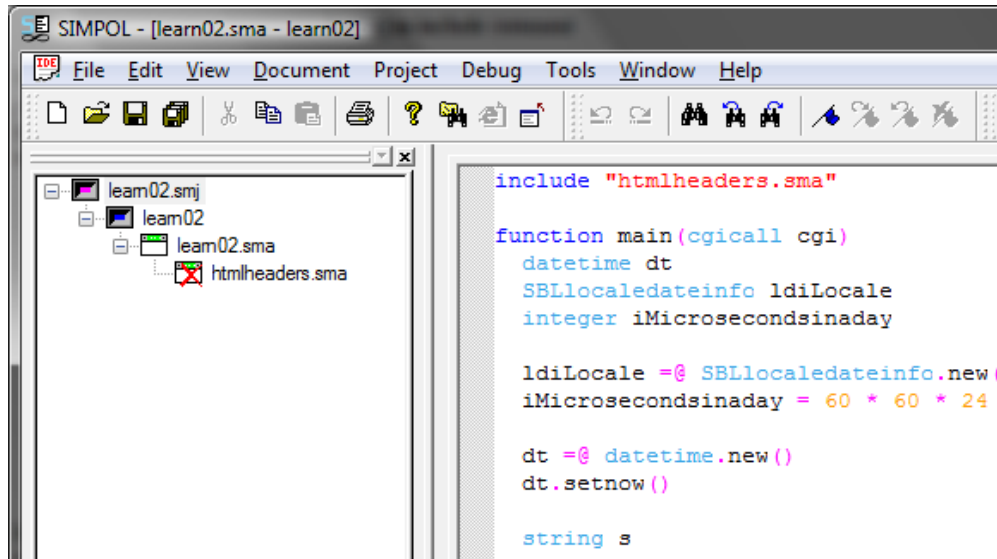
As a first step, we can save our project from the first chapter as a new project. To do so, open the first project in the IDE and then select the Save Project As ... item from the File menu.



Saving the project with a different name

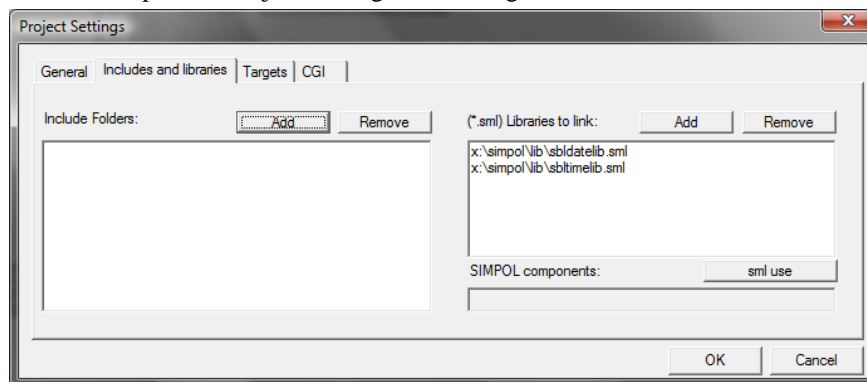
Then give the new project the name `learn02` and select the `tutorial` directory as the place to store it. This will create a new project called `learn02` with a main source file called `learn02.sma`.

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:



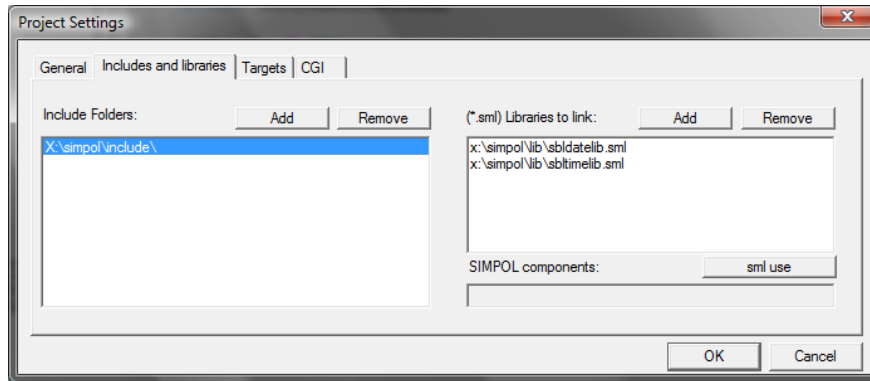
Adding an **include** statement

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.



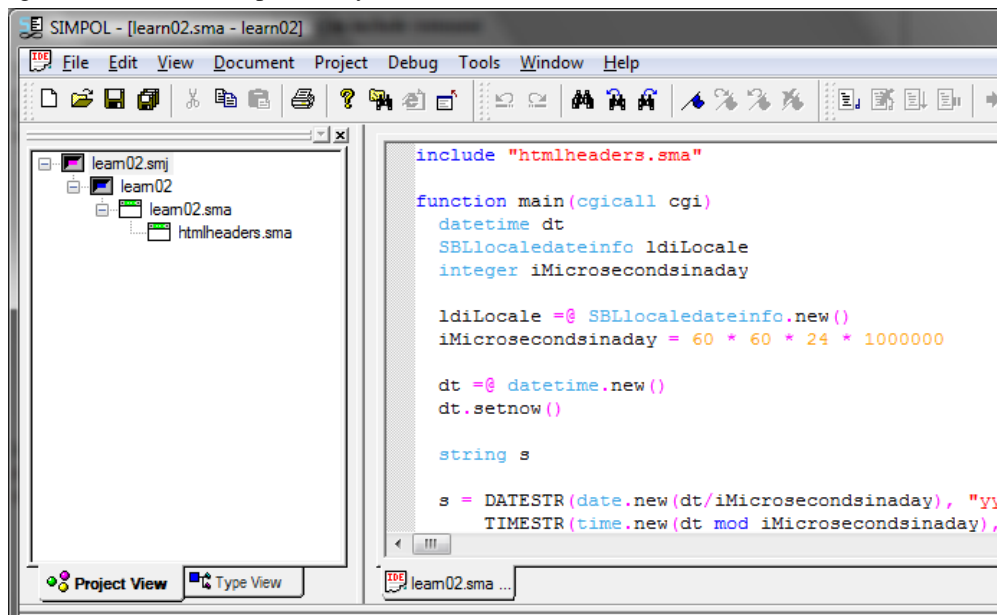
Adding an **include** path

Now click the Add button on the left side of the window and select the *include* directory that is directly below the root of the Superbase NG installation directory. When you return, the window should look like the one below:



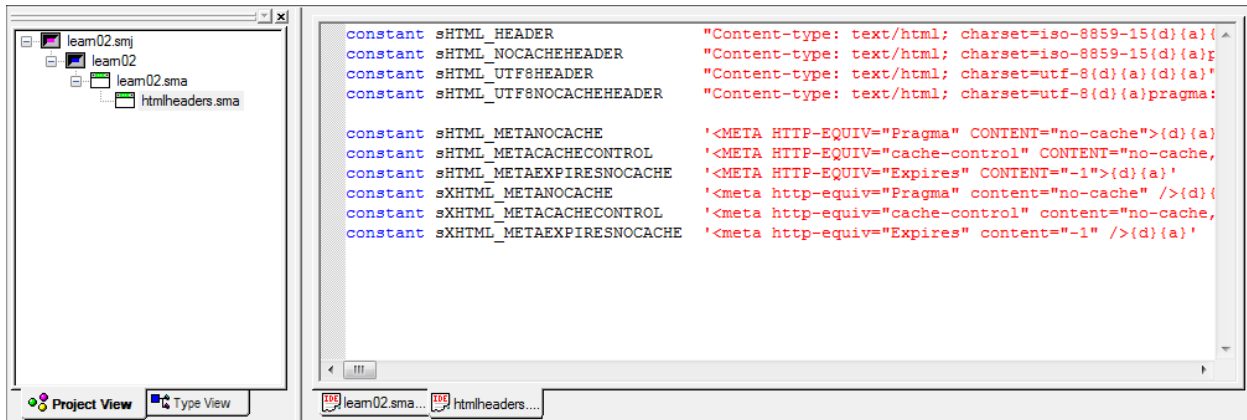
The include path has been added

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



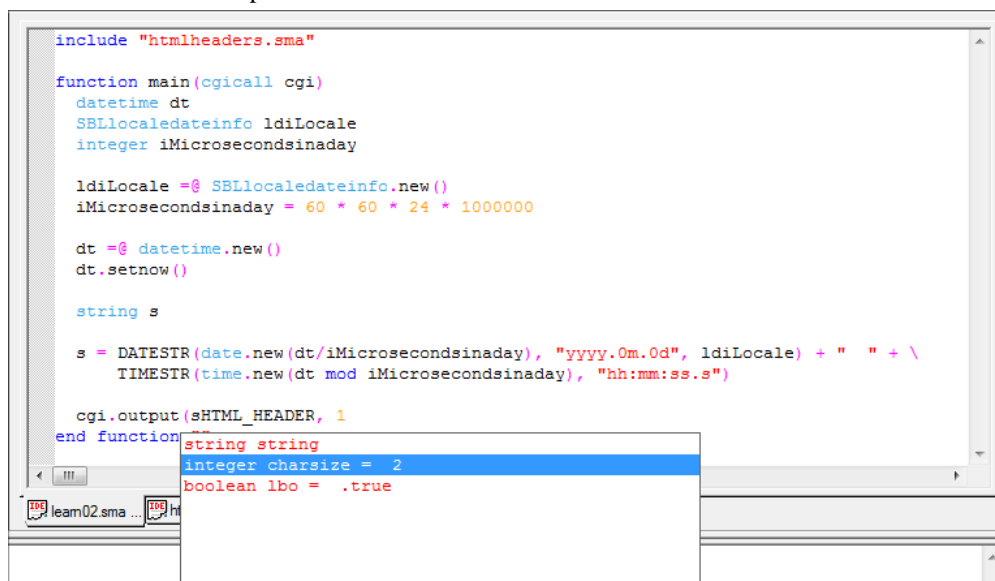
The icon now appears normally

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.



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.



Adding the code to output the header

Now complete the code as shown in the following picture. 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), "yyyy.0m.0d", ldiLocale) + " " + \
        TIMESTR(time.new(dt mod iMicrosecondsinaday), "hh:mm:ss.s")

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

```

The complete program code for our second program

Preparing the Web Server to Run SIMPOL Programs

We generally recommend that people pull down and use the free Apache web server, if not for deployment at least for development. Apache is the web server software that runs more than 64% of the world's web sites. It is included in most Linux distributions and as part of Apple's Macintosh OS-X operating system. There is also a Windows version. At Superbase we use the Windows version for development and testing and deploy on the Linux version. We also strongly recommend that if you intend to do web server development that you use Windows NT 4, Windows 2000, or Windows XP (at least until the Superbase NG IDE is available for other platforms). The reason for this is the heavy load placed on the operating system by the web server. A typical development environment for SIMPOL web server applications would consist of a web server running in the background, the IDE, a database server hosting database files via PPCS, either on the local machine or within the same LAN and a web browser or several for viewing the site in different browsers. On top of that there may be an HTML development tool like Macromedia's Dreamweaver and possibly a graphics package such as JASC's PaintShopPro. At any point in time there may be three or more instances of the IDE running. All of this produces a fairly large load on the older Windows 9x operating system. If you have no choice, make sure that you carefully read the section on configuring Apache below, since one of the settings can prevent everything from working at all.

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.

Getting and Installing Apache

As mentioned earlier, the Apache web server is a freely downloadable product which is part of the offering from the Apache Software Foundation (<http://www.apache.org>). To get the latest release (always recommended) retrieve it from <http://httpd.apache.org>. There is also a very recent release included on the distribution CD of Superbase NG in the Extras directory. Regardless of where you get it from, the first question you will have to answer is, "Which version?". Apache is available in three stable releases, versions 1.3.x, 2.0.x, and 2.2.x at the time of writing. Apache 2 is a reengineered version of the Apache 1.x web server with a major effort having been made to remove large pieces of the architecture from the web server kernel and place them into loadable modules. All of the following documentation will be based on the Apache 2 web server, but the configuration information is identical for both so there should be no problem regardless of which is installed. Downloading the web server is quite straightforward. Please use one of their mirrors closest to your location. If you can use the *.msi version, please download that, since it is considerably smaller than the *.exe version. Once you have downloaded it, run the package and accept all of the default values. The normal location is C:\Program Files\Apache Group\Apache2. One of the questions that you will be asked on a Windows NT, Windows 2000, Windows XP, Windows Vista, or Windows 7 operating system is whether to install as a service. We recommend that you answer this question with a no. The reason for this is that as of Windows Vista, services run in a separate Terminal Server instance and therefore cannot communicate with programs running on the desktop, which means that the debugging facility of the IDE would not work.



Note

If this is being installed on Vista or later, it is recommended to *not* install into the C:\Program Files directory. This directory is specially protected on Vista and trying to work with the configuration files of Apache within this directory may present considerable unnecessary problems.

Configuring 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 the web server has been successfully installed (no reboot of the computer is normally required), in the root directory of the Apache installation there will be a number of directories that concern us. These are:

- `cgi-bin`
- `conf`
- `error`
- `htdocs`
- `logs`

The `cgi-bin` directory is where the programs normally reside. The `conf` directory is where the configuration files can be found. The `htdocs` directory is the default location for your web site (to use a different one is outside of the

scope of this document, but see the section on virtual directories in the Apache documentation) and 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.

Of the above, the `error` directory is not really covered here, but is worth a mention. If a page is not found on a web site, typically an error number 404, "File not found" is returned. That doesn't help the user all that much and it can be far nicer to provide a replacement page that offers the user access to the site's search engine, etc. or at least a link to the home page. Most web servers allow for this and Apache is no exception. Look into the documentation on how to create your own error pages for the errors that might be worth handling yourself rather than leaving them to the web server defaults.

To configure Apache for use in general, there are only a few changes that need to be made to the configuration files that are created during installation. The first thing that you may need to change should only be changed if you are running on Windows 9x. That is the following parameter:

```
# WinNT MPM
# ThreadsPerChild: constant number of worker threads in the server
# process
# MaxRequestsPerChild: maximum number of requests a server process
# serves
<IfModule mpm_winnt.c>
ThreadsPerChild 250
MaxRequestsPerChild 0
</IfModule>
```

The number of threads per child should be changed to 6. For local development you won't need many anyway but if you try and use the default the web server may fail to even start. The next 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. If you don't have a DNS name for your workstation, you can get around this problem by assigning a name and domain in the `hosts` file which can be found in the `WINDOWS` directory on Windows 9x machines and otherwise in the `WINDOWS\SYSTEM32\DRIVERS\ETC` directory on Windows XP and in the `WINNT\SYSTEM32\DRIVERS\ETC` directory on Windows NT 4 and Windows 2000. In the worst case you can also always use the name `localhost` to identify the machine. In our examples we will use `localhost` in the browser URLs so as to be unambiguous.

```
# 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 www.mytestdomain.com:80
```

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

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this
# directory, but symbolic links and aliases may be used to point
# to other locations.
#
DocumentRoot "C:/Apache2/htdocs"
```

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:

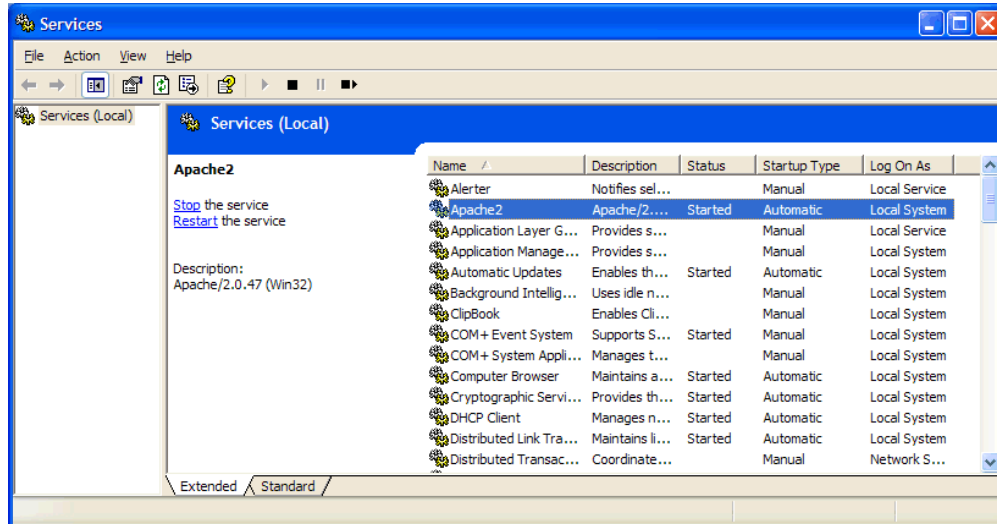
```
# ScriptAlias: This controls which directories contain server
# scripts. ScriptAliases are essentially the same as Aliases,
# except that documents in the realname directory are treated as
# applications and run by the server when requested rather than as
# documents sent to the client. The same rules about trailing "/"
# apply to ScriptAlias directives as to Alias.
#
ScriptAlias /cgi-bin/ "C:/Apache2/cgi-bin/"

#
# "C:/Apache2/cgi-bin" should be changed to wherever your
# ScriptAliased CGI directory exists, if you have that configured.
#
<Directory "C:/Apache2/cgi-bin">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>
```

No other parameters are required, but if you intend to run SIMPOL programs outside of the *cgi-bin* directory, then you will also need the *AddHandler* parameter. Now save the *httpd.conf* file. That completes the Apache portion of the configuration.

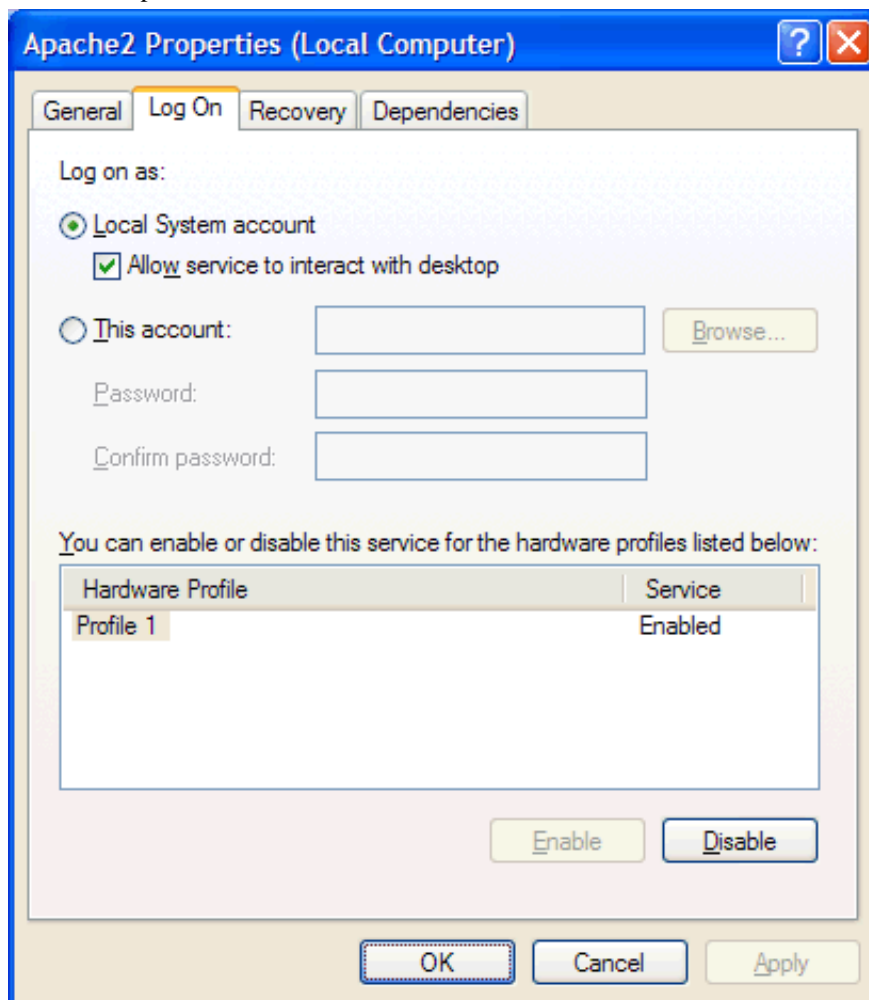
Configuring the Apache Service for Desktop Access

If you chose to install as a service (this *does not* work on Vista and later), to enable the ability to debug server program within the Superbase NG IDE, it is necessary to configure the service to interact with the desktop. Open the Control Panel, Select Administrative Tools and from within that group select the Services applet. When it starts the window should look something like the following:



The Services applet window

Double-click on the Apache2 service and on the second tab of the resulting window, checkmark the box entitled Allow service to interact with desktop as shown below:



The Apache2 properties window

Click OK to close the window and store the changes.



Note

Although there is a similar little box in the dialog in Vista, it does not do what it did before. It merely allows the service to show dialogs that can be reacted to by the user, in a special mode of operation, but it does not allow the service access to the session in which the user code is running, making it useless for our purposes on Vista and later.

Restarting the Apache Web Server

Once the configuration is complete, if you are running Apache as a service you will need to restart Apache. If you are using Apache 2, then in the Windows taskbar you will find an icon for the Apache control utility, which looks like a white circle containing a triangle on its point and a red feather off to the left upper side. Below is a picture to help you.



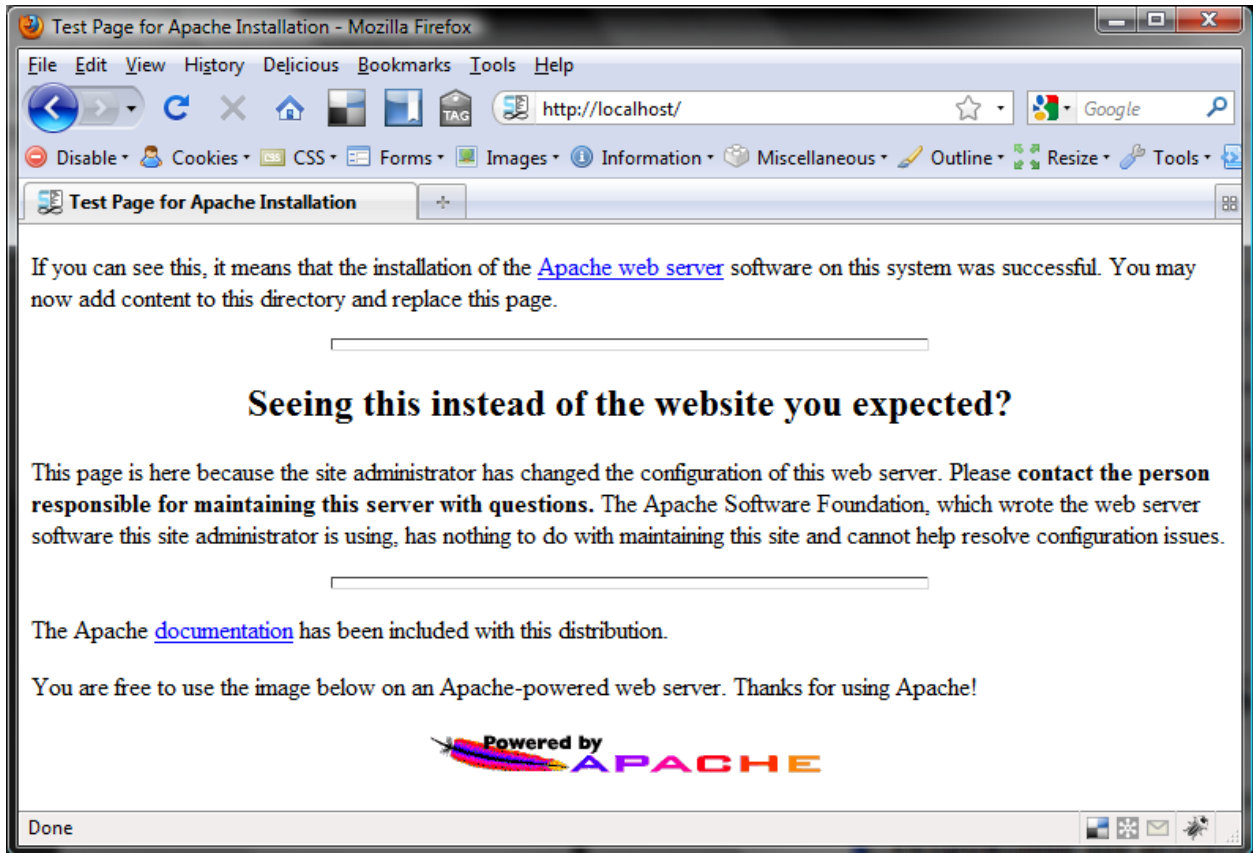
The Apache 2 control icon

Left-clicking on the icon will provide a menu with the options to Stop, Start, and Restart. Select the Restart option.

If you are using Apache 1.3.x then you will find menu items in the Windows Start menu for stopping and starting the web server. Stop the server and then start it again.

Alternatively, if you are running as a console application, you simply select the Start menu and navigate to the location that says Start Apache in Console. To close Apache later, just click the Close gadget.

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.

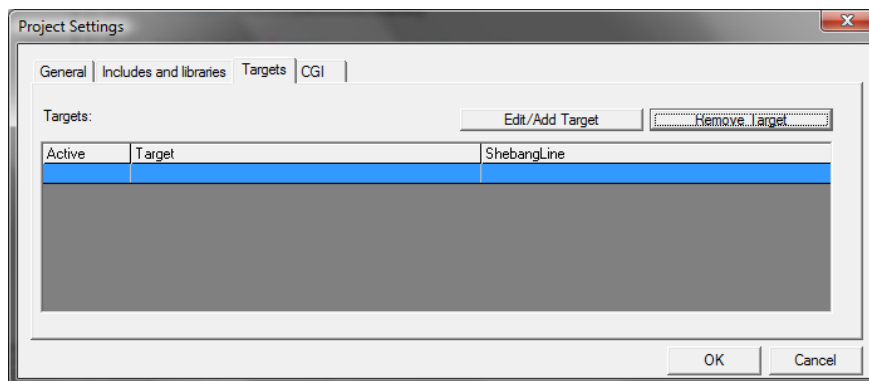


The Apache test home page

Once you ensure that the web server is working, you are ready to go on to the final part of this tutorial, actually running and debugging your web server programs.

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 labeled Targets, as shown in the picture below:



The Targets tab of the Project Settings window

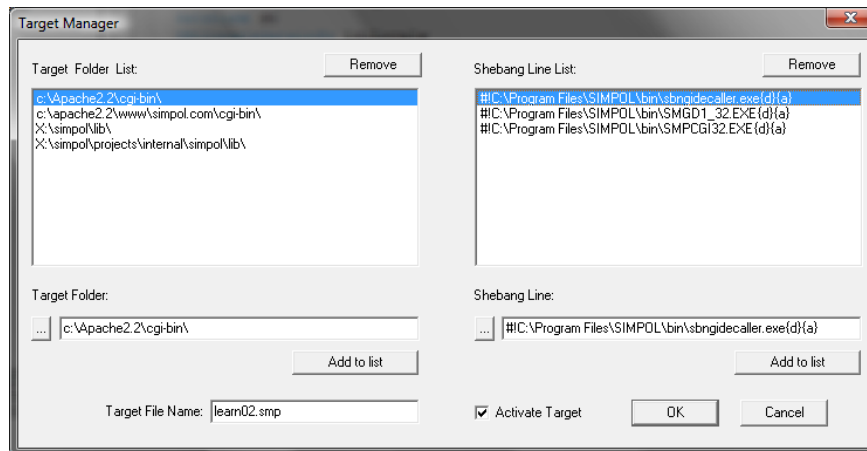
Click on the Edit/Add Target button. This displays the Target Manager window. On the left side of the window is a place to store common target directory names. On the right is a place to store common shebang lines.



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.

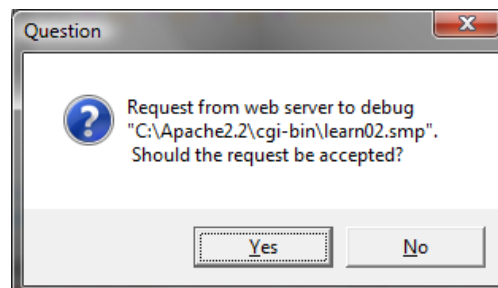
Create a target such as the one shown in the picture below. Make sure to use the correct path names for wherever you installed the Superbase NG product. You may wish to add the target directory and shebang lines to the lists since you will probably use them often. In this case we are creating a debug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



Creating our new target for `learn02.smp`

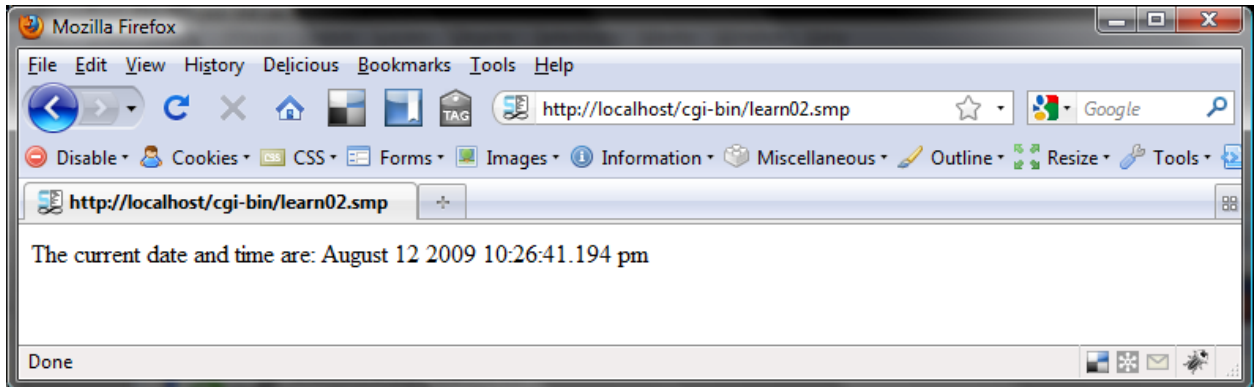
Don't forget to activate the target! Then click the OK button to create the target and then the OK button to save the changes. Finally, press **Ctrl+B** to rebuild the project and create the secondary target.

Now open a browser window if one is not still open and enter the URL `http://localhost/cgi-bin/learn02.smp` and press the ENTER key. The following picture should appear on your desktop if you are using the Superbase NG redirector program called `sbngidecaller.exe`.



Request to debug program

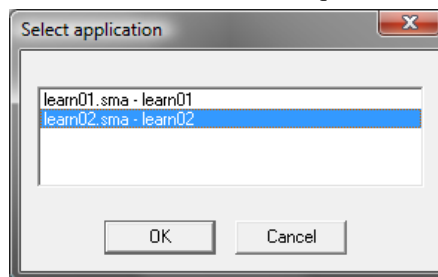
This message is from the IDE indicating that it has received a request to debug a program. If that program is not the current one in your IDE the current project will be closed and the project associated with the program to be debugged will be opened. At this point everything runs exactly the same as when debugging normally so we won't go into the details of that. Simply click on the OK button and then press **F5** to let the program just run through (it shouldn't have any errors in it this time, if it does you now know how to find them ...). The result should show up in your browser and look like the following:



Our program output in the browser

**Note**

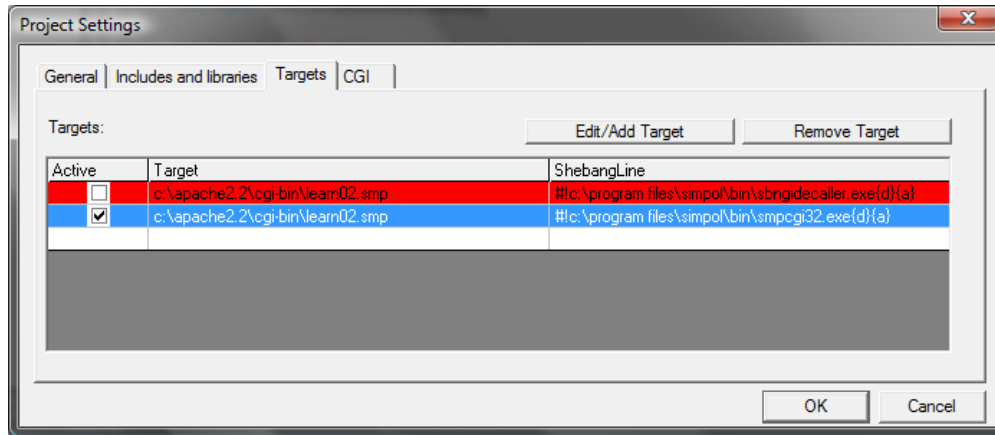
If more than one copy of the Superbase NG IDE are running concurrently then the first window that pops up will not be from the IDE about a request to debug, it will be from the `sbngidecaller.exe` program asking which copy of the IDE should receive the request to debug. That window looks like this:



Request to select an IDE for debugging

Select the appropriate entry and then the debugging request message will appear.

Once the program works as it should, we can go back into the settings for the project and add a non-debug target and deactivate the debug target. That way, if we ever need to work on the program again, our debug target is ready to be used. Once you change the target type, don't forget to rebuild the project!



Adding a non-debug target

Summary

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

Chapter 4. Debugging Into Library Source Code

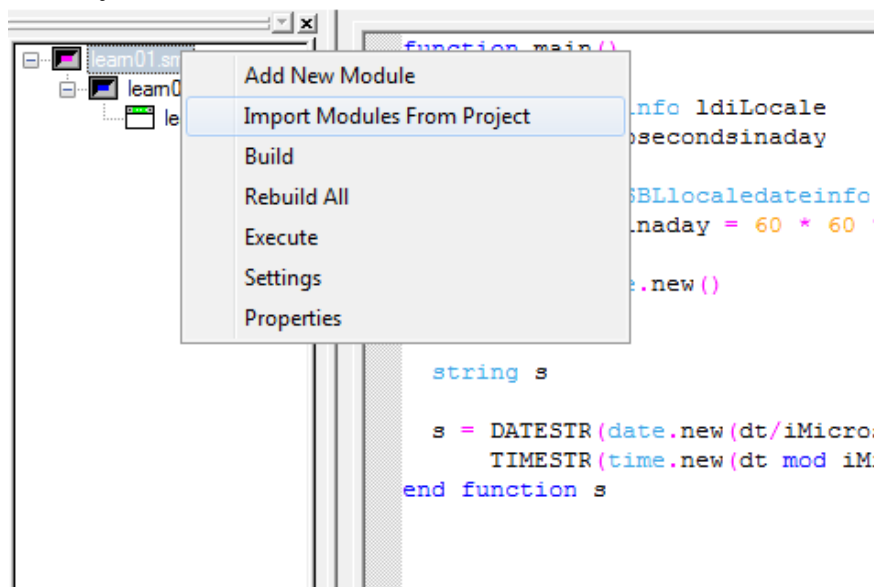
In the first chapter, we built our first basic program in SIMPOL and learned how to use the IDE to do various tasks, including basic debugging. In this chapter, we will learn how to debug into the source code of one of the supplied libraries.

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.

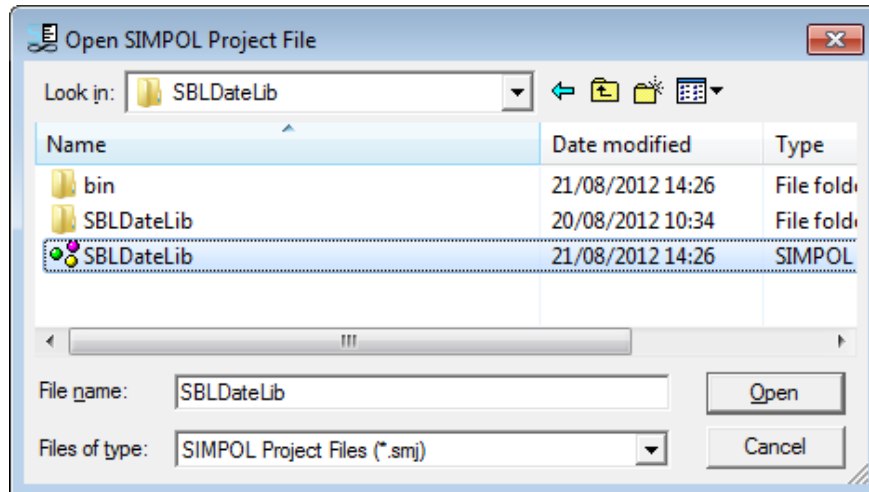
Adding Library Source Code

The first step in the process is to add the library source code to your project. We will use the source code from the initial project in Chapter 2, *Getting Started with the Superbase NG IDE*. To add the source code from another project to the current one, in the project tree view right-click the root node of the tree. From the resulting pop-up menu select, Import Modules From Project.



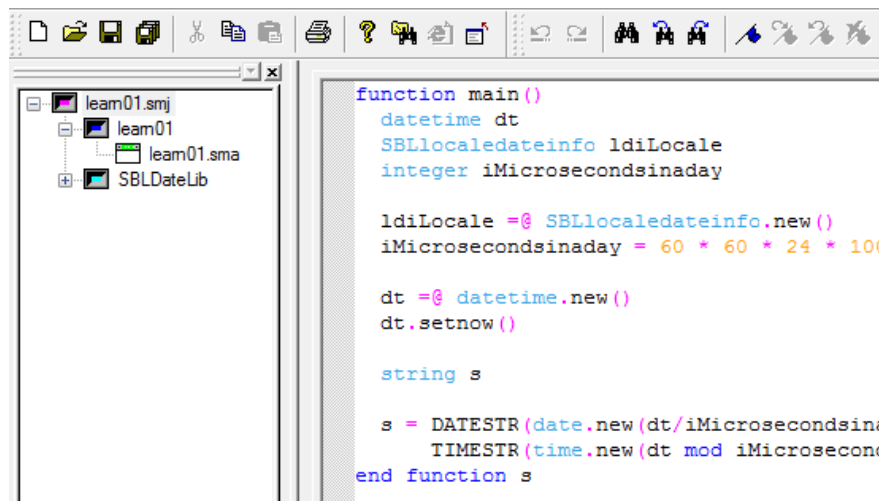
Project context menu

In the resulting file open dialog, select the project file for the library. In this case, we will use the `SBLDateLib.smp` project file.



Open project window

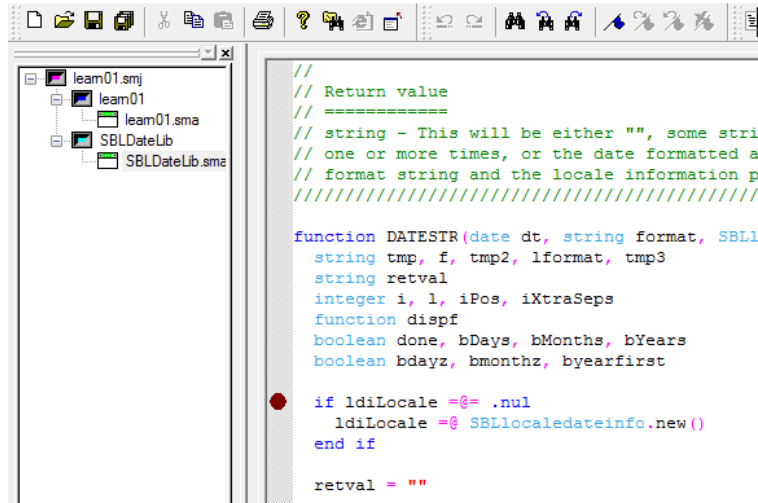
This will add the library project into the current project as a module. Projects loaded in this way cannot be modified, the source code is read-only. Once it has been added, the IDE will retrieve the function and type information, and add it as a module to the tree.



Project after importing a module for debugging

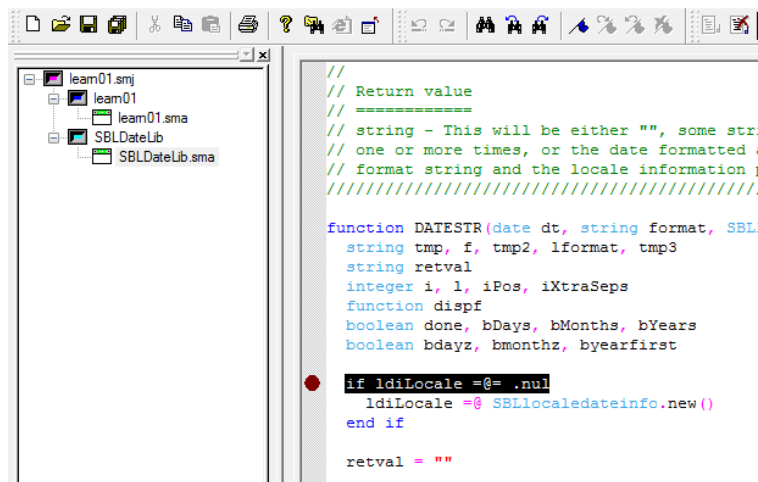
Debugging Library Source Code

Now that the project code has been imported, if we now expand the entry for SBLDateLib and double-click on the SBLDateLib.sma item, that will open the source file in the editor. Next we will place a break point at the beginning of the DATESTR () function, so we can debug inside the call to the library function.



Setting a break point in a library function

As we did in the first chapter, press **F4** to enter debugging. Then press **F5** to run the program. Almost immediately the program execution should halt with the line containing the break point highlighted, as shown below:



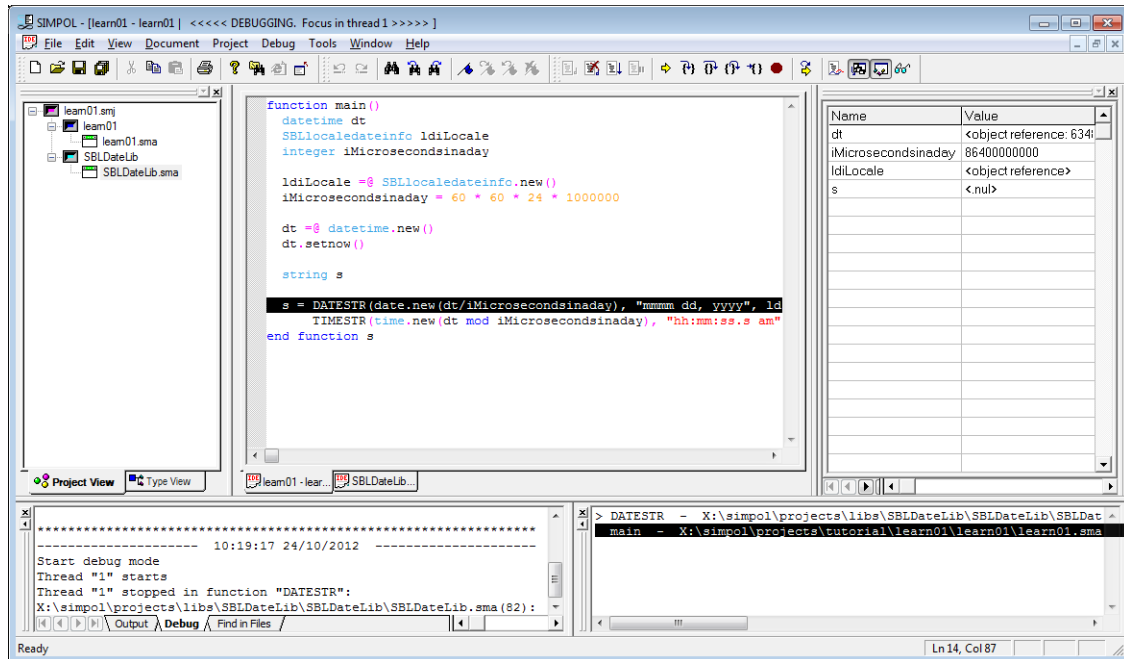
Code execution halted at the break point

At this point, we can start single-stepping through the code, 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

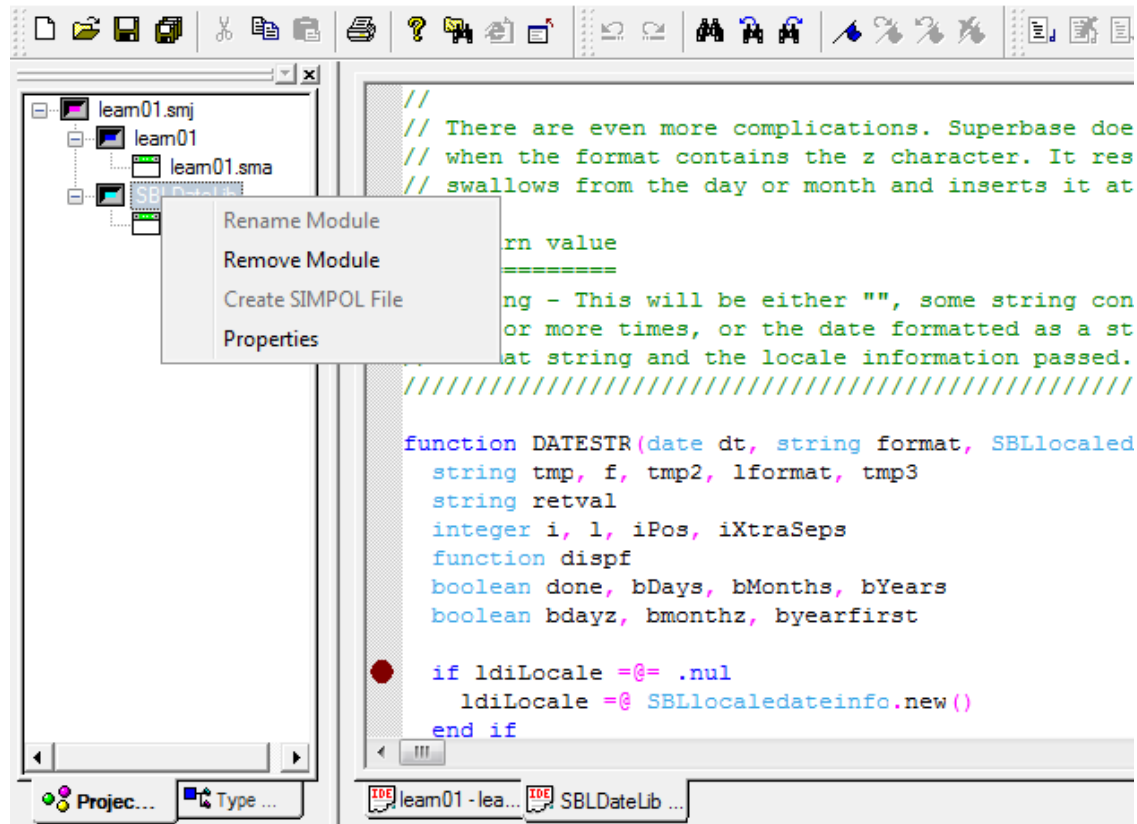
If you are debugging code that is working with a multi-threaded system, such as that provided by the SIMPOL Application Framework, then you may need to switch to the correct thread before any debugging commands will work or the variables are shown in the list. To do this select **Debug** → **Thread Manager**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



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.



Removing the imported source code module

Summary

In this chapter 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!