



CODE **N**EO™

**APPLICATION
SERVER**
and
**DEVELOPMENT
TOOLS**

CODE:NEO Getting Started Manual

INTRODUCTION;	3
WEB TECHNOLOGY BASICS;	6
CODE:NEO BASICS;	10
CREATE RIVET OR JOINT	11
Creating a rivet.....	11
CONVERTING A RIVET TO A SOURCE JOINT	12
COMPILE SOURCE JOINT TO BINARY JOINT	12
CALLING THE CODE:NEO BINARY JOINT	13
CREATING A CODE:NEO JOINT;.....	13
GETTING STARTED WITH THE WDE	14
GETTING STARTED WITH VISUAL STUDIO	27
DEBUGGING CODE:NEO APPLICATIONS;	34
DEBUGGING BASICS.....	35
On windows	35
On unix.....	35
CHOOSING THE COMPILATION METHOD TO INCLUDE DEBUG SUPPORT	35
ON WINDOWS	35
ON UNIX	36
REAL TIME DEBUGGING	36
REAL TIME DEBUGGING WITH GDB	36
REAL TIME DEBUGGING WITH KGDB	37
REAL TIME DEBUGGING WITH MICROSOFT VISUAL DEBUGGER	37
REAL TIME DEBUGGING WITH CODE:NEO DEBUG TRACE WINDOW	41
ON WINDOWS	42
ON UNIX	42
DEBUG INCLUSION.....	42
DEBUG USING LOGGING	43
USING COOKIES WITH CODE:NEO	44

Introduction;

Welcome to CODE:NEO, the premier tool for creating Internet Applications with C++. CODE:NEO was developed with the purpose of creating the most powerful application server environment possible, to that end we decided to utilize C++ because of it's vast advantages for building powerful high speed applications.

What CODE:NEO adds to the powerful C++ language is the power to integrate the C++ into HTML, and other markup languages in such a way that the file can be converted into a source code file that can be compiled with the developer's native C++ compiler.

The output of this process is a binary file that can be loaded into memory and then executed natively by the processor by changing the point of execution into that loaded binary object. This has many advantages over similar technologies, which rely on software execution of the developers program instead of hardware execution.

Likely your knowledge of technology and the advantages of C++ lead you to purchase CODE:NEO. Perhaps your existing commitment to C++ lead you to CODE:NEO because you wish to directly use code that you had already written, or perhaps your frustration with other web development tools prompted you to reach out to the most powerful tools that you could find.

Regardless of how you arrived at CODE:NEO we hope that you find our product very useful, easy to use and difficulty free. The CODE:NEO development team is always developing and improving the product and looks to developers and their suggestions to motivate our enhancements to the system. Special requests are greatly appreciated and welcomed and hopefully we can implement them in new versions as soon as possible.

CODE:NEO is also architected to be as multi platform as possible and we strive to build distributions to as many platforms as are requested. Ports have been made because of single requests and will always be considered. If you wish CODE:NEO were on another platform of your choosing please let us know and hopefully the CODE:NEO development team can bring the CODE:NEO product to your desired platform soon.

At the time of this writing, CODE:NEO is available on Solaris, Linux, Windows 2000, Windows XP, Windows NT and Windows 9x. Also at the time of writing CODE:NEO was compatible and extensively tested on Apache and Microsoft Internet Information Server web servers.

CODE:NEO should also work on other ISAPI compliant web server, if you have CODE:NEO working on any other web server please contact us so that we can list it here.

CODE:NEO and our development tools have been built specifically for GNU's gcc also know as egcs, Microsoft Visual C++ and has been reported to work with Borland C++.

As a company dedicated to providing the best development tools available the CODE:NEO development team strives to fully support our product and our developers, in an effort to provide this we provide email and phone support for our customers. 24x7 emergency support is also available on a request basis, if you feel you would need this type of support please contact a customer service representative so they may provide you with that contact information.

This getting started guide is designed to bring a C++ developer to a level in which he/she may begin development with the CODE:NEO product, we expect that only an hour should be necessary for the developer to begin developing with CODE:NEO.

The beginning of the getting started manual is dedicated to giving a brief overview of the CODE:NEO development tool and application server, while the remainder of the manual is dedicated to giving the developer a more detailed look at the tools in the CODE:NEO package.

For most developers only a brief overview of the material should be necessary to being and then the developer may return to this manual if they wish to exploit a particular advantage of CODE:NEO.

All of the source code examples shown are available for download from our site.

Except as noted **Bold Courier** identifies literal source code, in some sections differences for each platform are listed and are preceded by a title bar for the particular platform. UNIX is used to represent and UNIX type operating system such as Solaris or Linux, Windows covers all Win32 compatible platforms. Sections that are relevant only to a particular platform market with the specific platform name.

As with any human work, errors are possible and likely, we encourage people to point out any errors to this manual at bugs@codeneo.com.

Other contact information you should be aware of are, support@codeneo.com for support with our product, sales@rivar.com for sales information and vince@rivar.com who is the author of this manual and inventor of CODE:NEO. I always look forward to hearing from people using or considering CODE:NEO for their development projects.

Our main phone number, useful for any reason is 727-461-4555. Our website for the product is <http://codeneo.com> and our company website is <http://rivar.com>, a wealth of information can be found at those sites about our products, services and company information.

You may mail us at;

Rivar Technologies, Inc.
Suite 311
51 S Main Street
Clearwater, Florida 33765

Welcome to the CODE:NEO product, we hope to be your best asset for application development and look forward to any comments you have.

Web Technology basics;

As CODE:NEO allows C++ developers to begin rapidly developing web applications some C++ developers may have limited or no web development experience. To assist those developers we have included this brief section to get the developers started.

If you are already experienced with web development feel free to skip to CODE:NEO basics. This section is only intended to lay groundwork for developers so they can turn to other resources about web development.

Web technology relies upon users connected to networks to utilize a client application called a web browser. The web browsers makes 'HTTP' connections to a web server to obtain generally files it can display. One of the most common types of files it returns is a HTML file.

'HTTP' is a TCP/IP protocol, which lays out the method of communication between the web server and the web client (or rather browser). 'HTTP' communications generally happens on well-known PORT 80.

The HTML file is a human readable source code file for displaying a page the user can interact with including pictures, text, buttons, selectors, etc. More advanced HTML files even include JavaScript and Java Applets and Microsoft ActiveX to improve functionality.

HTML language is a markup language, which means that areas have been 'marked up' or denoted by symbols, which represent the beginning and end. For example is you wanted to display the message 'hello world' with HTML you would simple type, '**hello world**', but if you wanted the world to appear in bold such as 'hello **world**' you would simply mark the beginning of the bold area with '' and the end of the bold are by '' such as '**hello world**' in your html file.

HTML files generally have the extension '.html' or '.htm'.

The structure of an HTML file allows for several individual sections marked 'head', 'title' and 'body'.

A simple HTML would look like.

```
<HTML>
<BODY>
hello <B>world</B>
</BODY>
</HTML>
```

Where the 'html' marked section is a formality to the specification of an html document.

You may name the web page by adding a 'title' section such as;

```
<HTML>
<TITLE>
hello world web page
</TITLE>
<BODY>
hello <B>world</B>
</BODY>
</HTML>
```

Tags may be upper case or lower case. So that;

```
<html>
<body>
hello <b>world</b>
</body>
</HTML>
```

Has the exact same meaning and looks like;

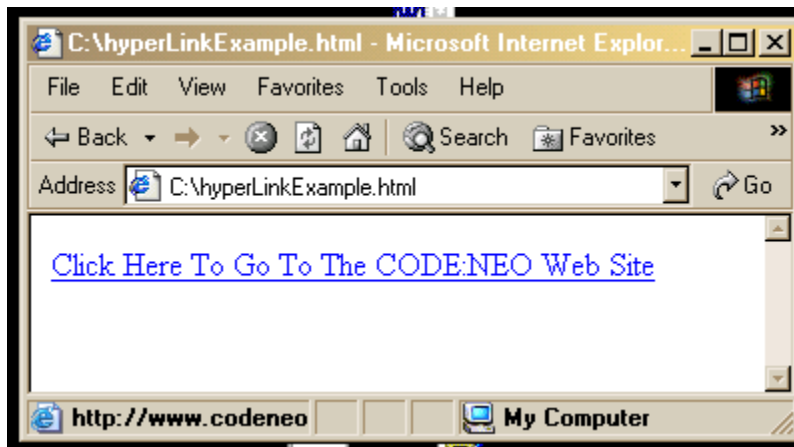


A web page can be accessed either directly from the users hard drive or through a web server. When accessed directly on the users hard drive the path name may be used. But when accessed through a web server the protocol, server name, directory and path must be specified like, 'http://servername/directy/file.html'.

Links from one file to another may be created with the 'href' tag, for example the following code makes a link the user may click on in the web browser to go to the CODE:NEO web site.

```
<html>
<body>
<a href="http://www.codeneo.com">Click Here To Go To
The CODE:NEO Web Site</a>
</body>
</HTML>
```

Which looks like;

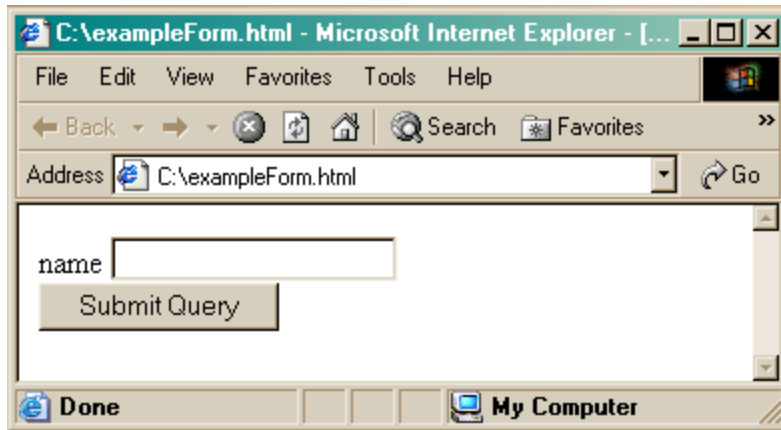


The easiest way to collect information from a user is to collect it via a HTML form. HTML forms specify the types and names of information to be collected.

A simple form to collect a persons name could be.

```
<html>
<body>
<form>
name <input name=name><br>
<input type=submit>
</form>
</body>
</HTML>
```

And looks like;



The '`<form>`' tags denote where data will be collected.

Each '`input`' tag denotes a new URL argument, each piece of data being collected from the user.

The information that the user fills into the name area is passed to the web server through either one of two methods. GET or POST. GET method, which is the default, passes the information on the URL line, better known as the 'address'. It separates arguments, individual pieces of information being sent or 'submitted' by the form with the '&' character and separates the query string, all the arguments, from the page name by a '?'. The information is separated by the name of the argument by the '=' character.

So if a user was to put the name 'jack' in the form above, the resultant URL that would be sent to the web server would be 'exampleForm.html?name=jack'.

POST method allows the information to be sent as the payload of the web page request, thereby obscuring it from the view of the web browser user.

We will go into greater details about interaction with the user later in this guide.

In the example form you will also notice the tag, '`
`', another of the few tags that do not mark the beginning and end of a section but a literal place in the web page, marks where the line breaks between the form field and submit button.

CODE:NEO Basics;

CODE:NEO works by allowing developers to create binary files that can be executed through our application server called CODE:NEO applications.

To build a CODE:NEO application you must create source files that when compiled, results in a binary executable, which the CODE:NEO application server can load into memory and execute.

The binary files are depending on platform either a dynamic shared object (on unix) or a dynamic link library.

The binary file is called for simplicity a CODE:NEO '*binary joint*', as in the connection between the processor and application. The extension for this file is generically '.CnBin'.

The specification for that '*binary joint*' is implemented in a CODE:NEO '*source joint*' and generally bears the '.cpp' extension.

Where the *joint* is the connection to the processor a CODE:NEO '*rivet*' ties C++ to another language such as HTML and generally bears the extension '.CnSrc'.

For HTTP (Web Server) applications the CODE:NEO web application server plugs into the HTTP (Web Server) through either an ISAPI Extension or an Apache module.

When a request comes in the web server recognizes the extension '.CnBin' (a CODE:NEO '*binary joint*') and passes control to the CODE:NEO web application server.

In short the CODE:NEO application server creates a request object and calls an entry point in the '*binary joint*' called '**main**' which creates a 'response' object.

From within that '**main**' method of a '*joint*' developers execute commands appropriate to fulfilling a server request. After the call to '**main**' returns execution back to the CODE:NEO application server, the application server interrogates the response object and responds to its client the appropriate response.

In the case of an HTTP application the developer would likely call '**swritef**' to write HTML to the response or '**getArgument**' to retrieve URL arguments into variables, i.e. to gather information from the web browser user.

Create rivet or joint

The first step in creating a CODE:NEO application is creating the source files that will be used to create a binary executable that the application server can run.

Two source file options exist that you can create and many mechanisms exist to assist you in creating those text source files. You may use standard text editors like VI, notepad, etc. to create the files for processing or you can use more sophisticated tools like the Web Development Environment.

CREATING A RIVET

To create a rivet you type in HTML and C++ using tags to mark where each type of code is. HTML is the default language so for example;

```
<html><body>hello world</body></html>
```

is a valid CODE:NEO rivet. While it does not have any C++ code or functionality it will still convert into a binary executable the CODE:NEO application server could execute. It simply would result in output the same as above.

A more complex example would be;

```
<html><body>Count to ten<br><%Cn>  
for(int x=1;x<=10;x++)swritef("%d<br>",x);  
</Cn%></body></html>
```

after conversion to a source joint, then compilation to a binary joint would output "Count to ten", then 1 through 10, each on its own line.

CODE:NEO tags separate the C++ and HTML. The '**<%Cn>**' tag demarks the beginning of C++, it is called the CODE:NEO opening tag and the '**</Cn%>**' demarks the ending of C++ which is called the CODE:NEO close tag.

Tags may have one of two scopes, global and inline. Scope is by default inline and can be changed to global by specifying it as, '**<%Cn scope="global">**'. Global sections can be closed with the standard CODE:NEO close tag.

The scope controls how the CODE:NEO development tool places code in the resultant source joint file. Code blocks, which are inline, are placed in the order that they are found in the rivet source file. Code blocks, which are global, are placed in the order they are found in the beginning of the source joint in C++ global space.

An example of this would be;

```
<%Cn scope="global">int x;</Cn%>  
<html><body>Count to ten<br><%Cn>  
for (x=1;x<=10;x++) swritef ("%d<br>", x) ;  
</Cn%></body></html>
```

Converting a rivet to a source joint

Since the developer has created a *'rivet'* it must be converted into a source joint before it can be useful. To do this he may use any mechanism that triggers the CODE:NEO development tool to convert the rivet into a source joint.

They may use the CnGen command directly as seen in the visual studio project or example make files, or detailed in the reference documentation.

They may click on the build command within the CODE:NEO web development environment.

They may trigger the build process for Visual C++, which automates the calling of the CnGen command.

They may execute the make command to begin the build process.

Compile source joint to binary joint

The source joint must be compiled and linked into a binary object that can be executed by the application server. To do this the developer uses his native compiler, which reads in the source joint and outputs binary code.

After the joint has been outputted to object code, the linker then integrates the CODE:NEO libraries, system libraries, developer's custom libraries and outputs a complete binary executable in the form of either a DSO or DLL for that developers platform.

Examples of compilers to be used include Microsoft Visual C++, GNU GCC, etc.

In addition to binary libraries and c++ header files, CODE:NEO also provides example make files and a Visual Studio project wizard to simplify development with CODE:NEO.

Calling the CODE:NEO binary joint

If CODE:NEO is installed and configured and your web server is configured, you may simply transfer your binary joint into your web server directory and call it by file name. Remember that CODE:NEO binary application files generally end in the extension '.CnBin'.

Creating a CODE:NEO joint;

In some circumstances programmers may wish to create joints directly. To do this instead of creating a file, which contains C++ and HTML, they must create a pure C++ source file, which conforms to the CODE:NEO joint specifications.

The following is an example of a CODE:NEO joint hello world program;

```
#include <CnJoint.h>

#include <CnHtmlServer.h>
using namespace CodeNeo;

class CnJoint : public CodeNeo::CnHtmlServer
{
public:
    CnJoint(CodeNeo::CnRequest *pCnRequest) :
CodeNeo::CnHtmlServer(pCnRequest) {}
    void main();

private:
};

CNJOINT_ENTRYPOINT(CnJoint)

void CnJoint::main()
{
    write("<html><body>Hello
World</body></html>\n");
}
```

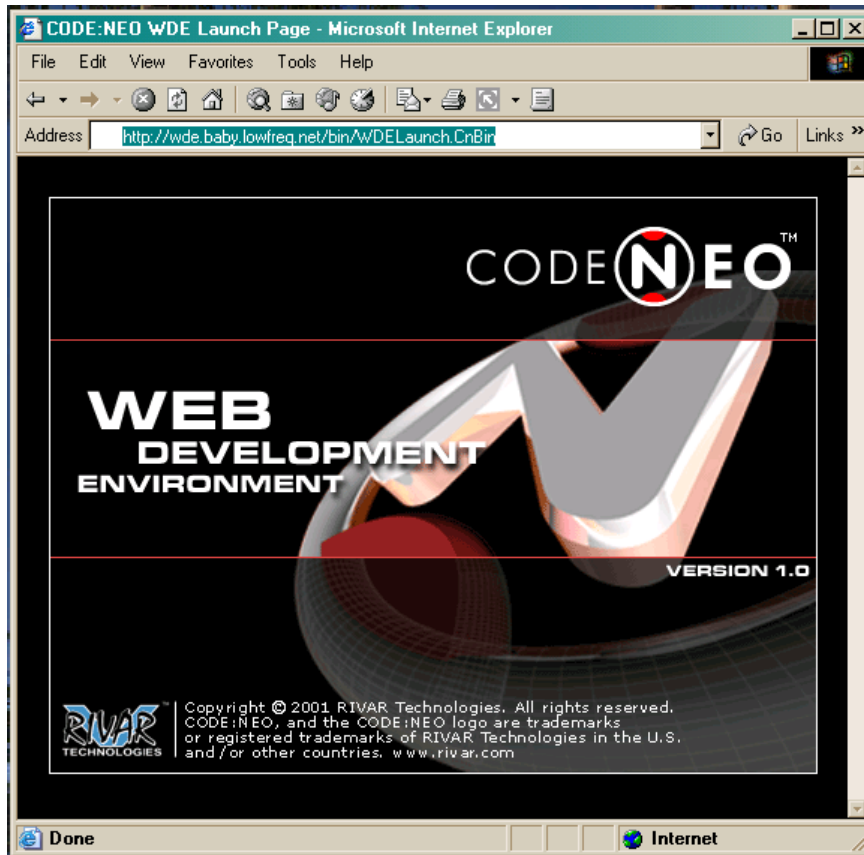
When compiled this CODE:NEO application will output the html;

```
<html><body>Hello World</body></html>
```

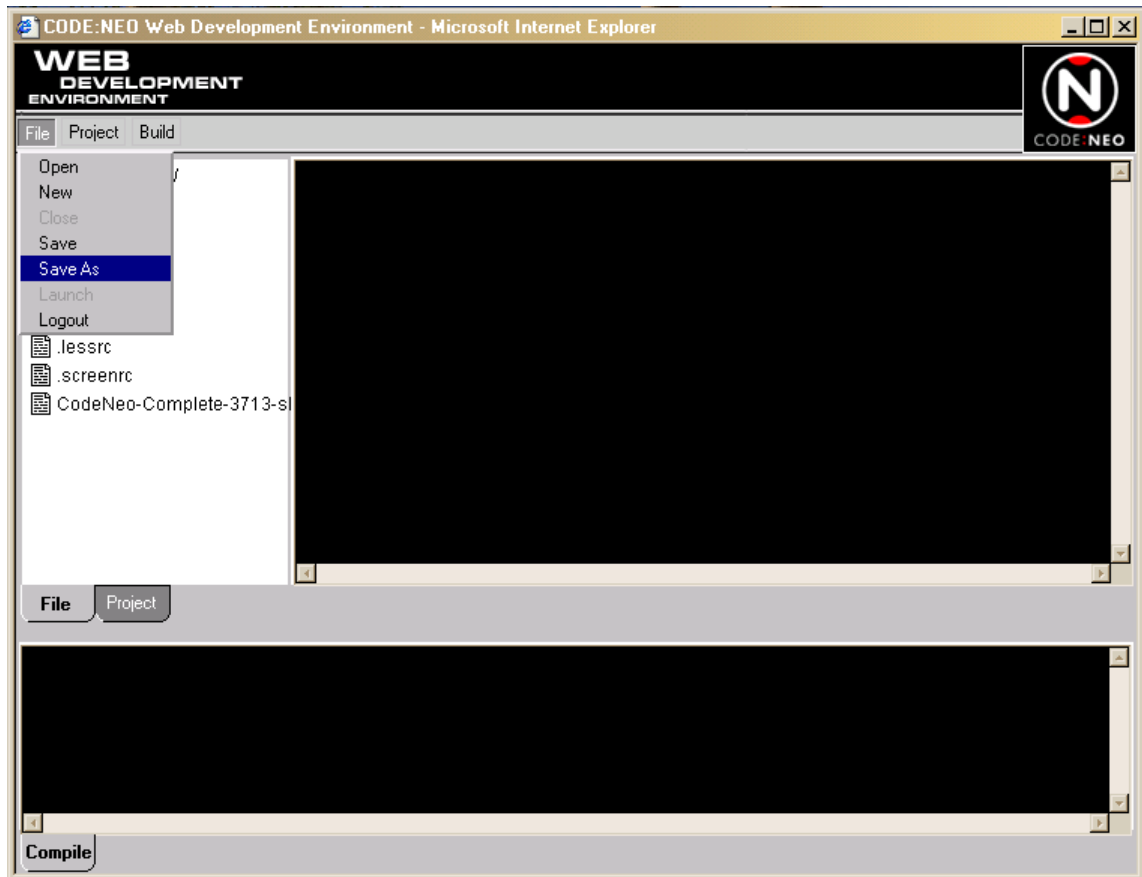
This includes infrastructure that defines the resources the joint has available. The file then declares that this file will use the namespace CodeNeo, which the CODE:NEO API is coded to be present in. It then defines an object, which extends CnHtmlServer, which is the base class for HTML joints. A macro then hooks the newly defined joint to an entry point the CODE:NEO application server can call. Then we implement a main function that will be called after the object is created and prepared by the application server.

Getting Started With The WDE

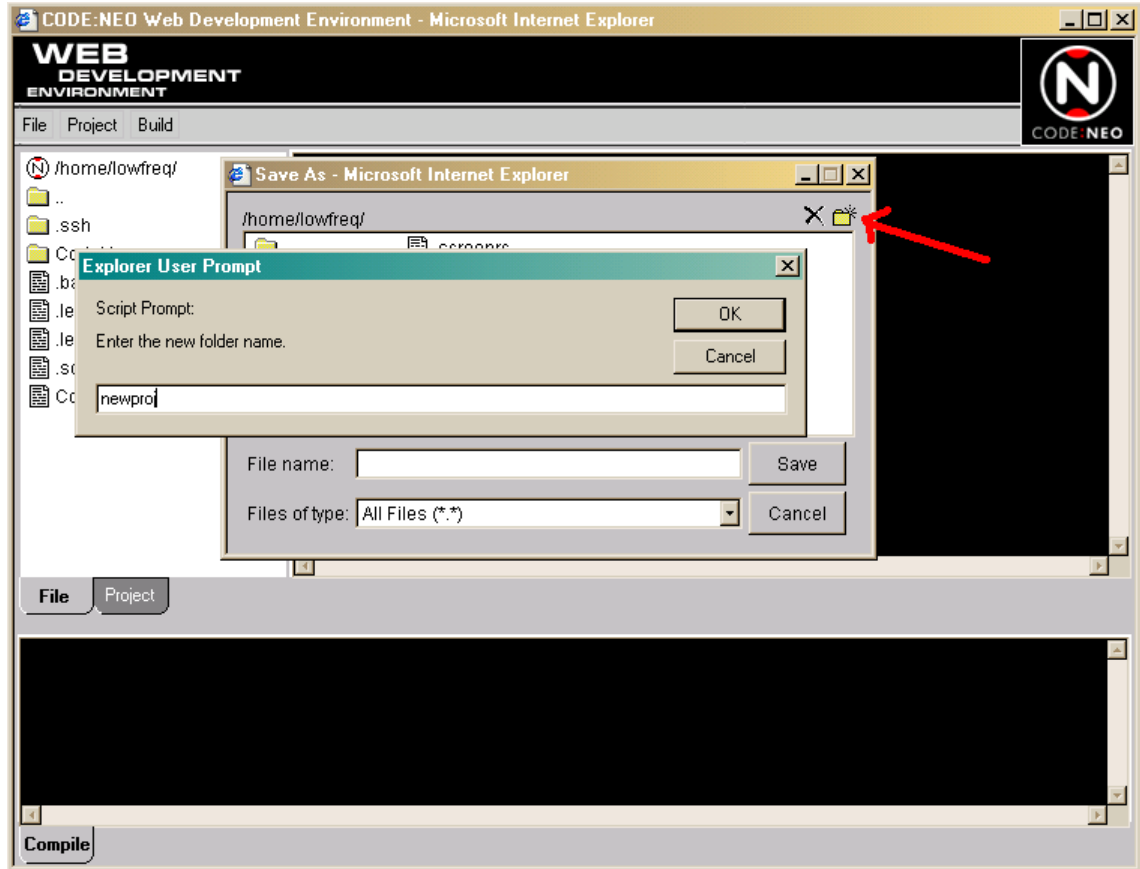
1. launch the WDE, the system will ask for your username and password, then this launch window will refresh with a WDE window that opens...



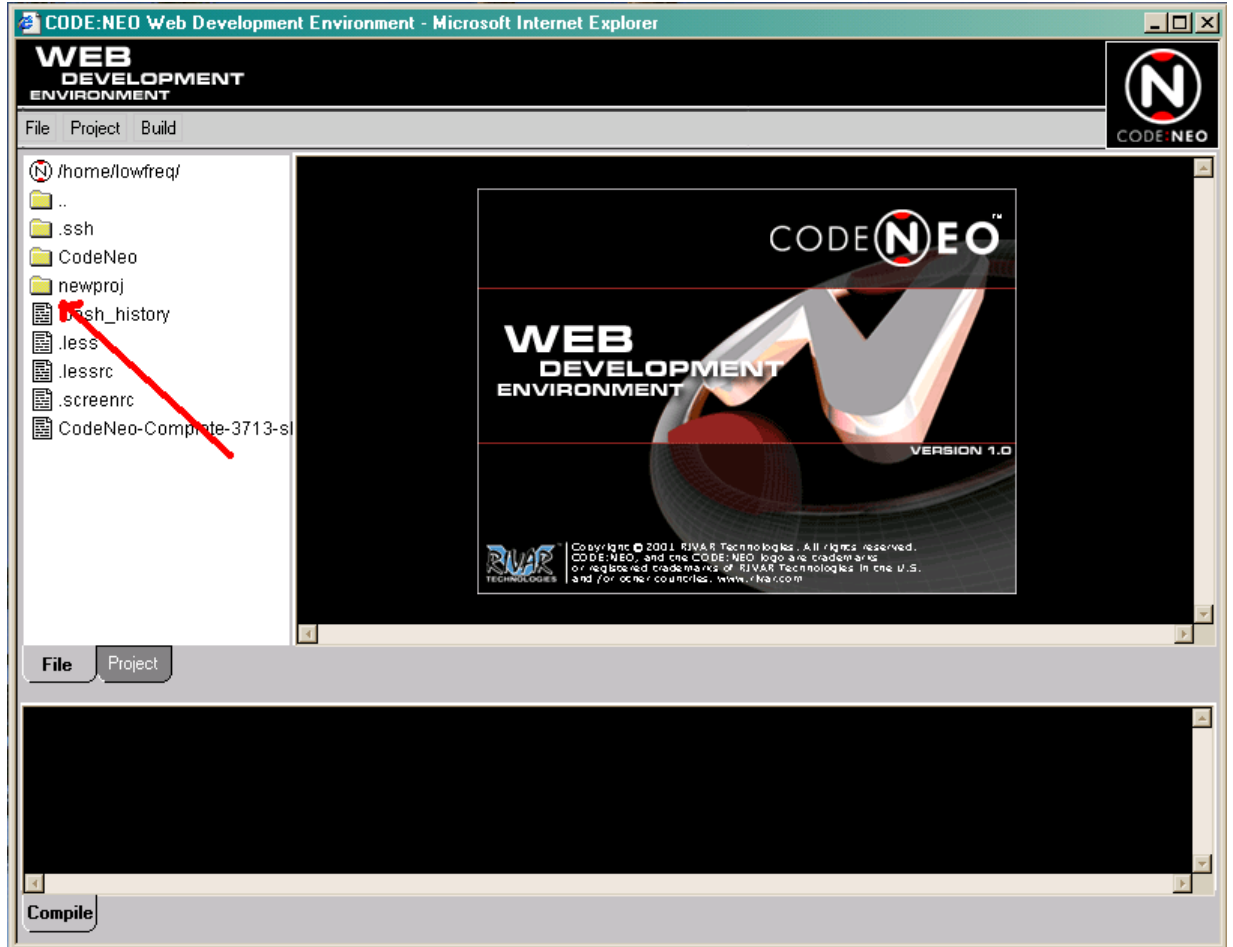
2. go to save as to create a 'newproj' directory



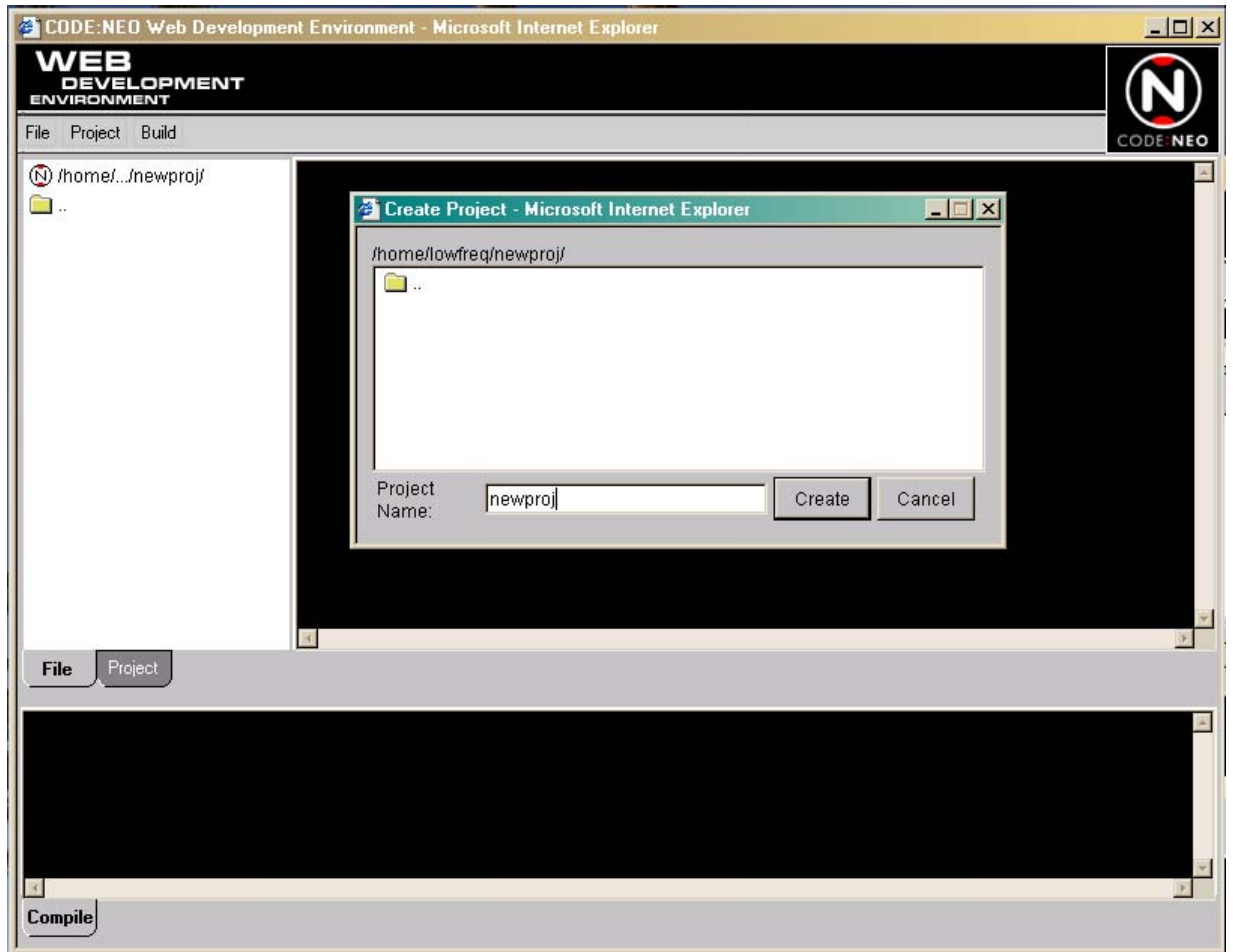
3. click on new folder to create a new folder, input project name, click ok



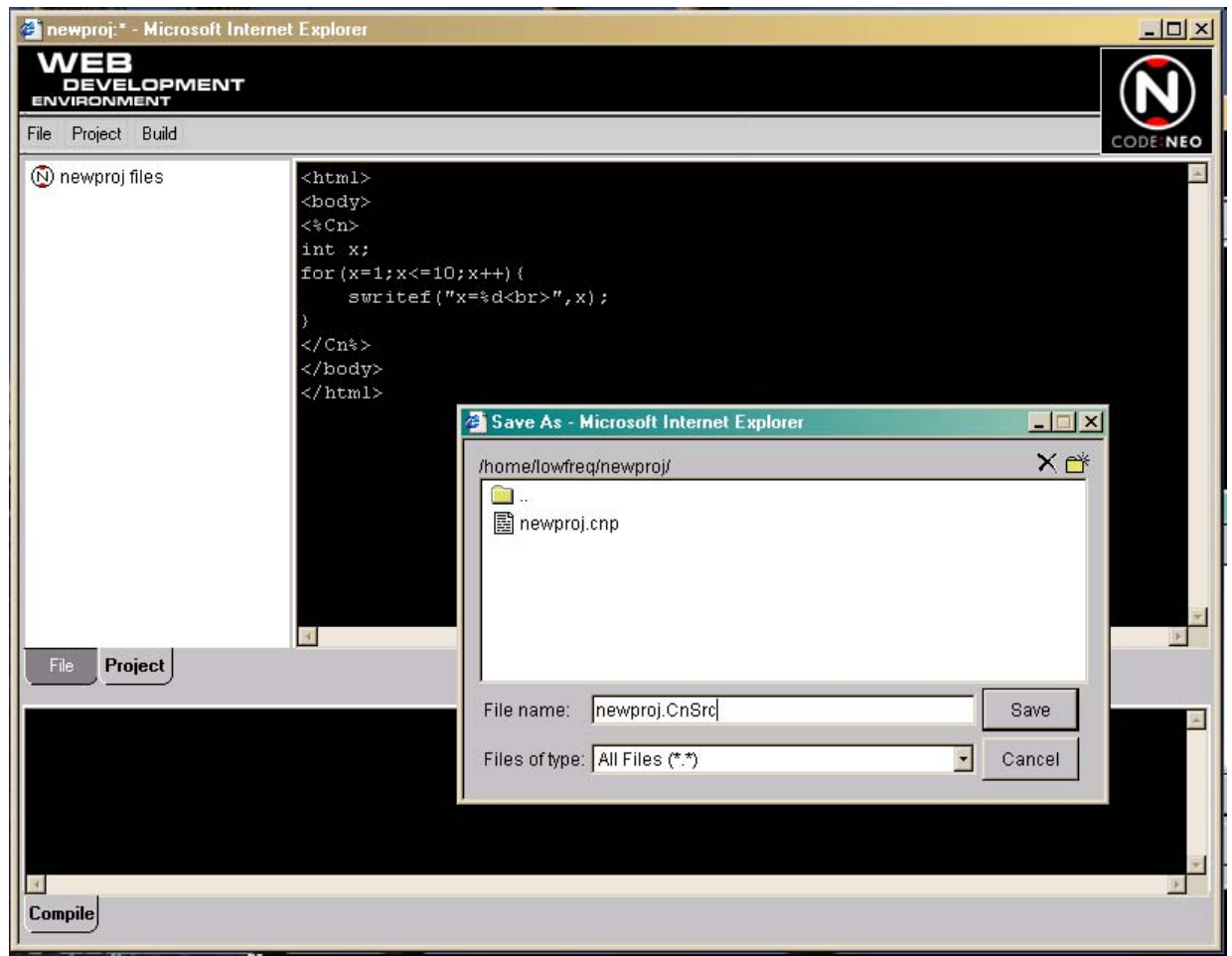
4. click cancel and then f5 to refresh to see new folder



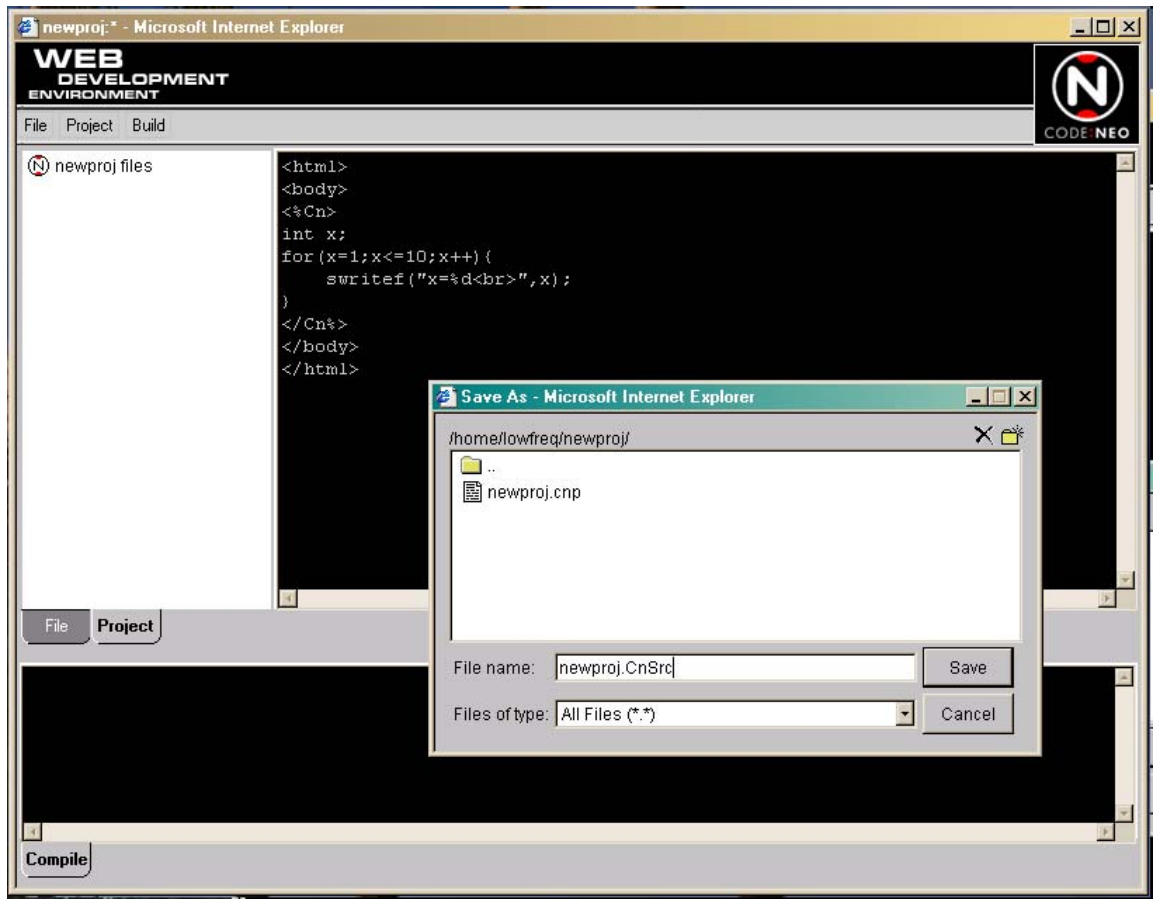
5. double click folder to enter folder, then click project create and fill in project name



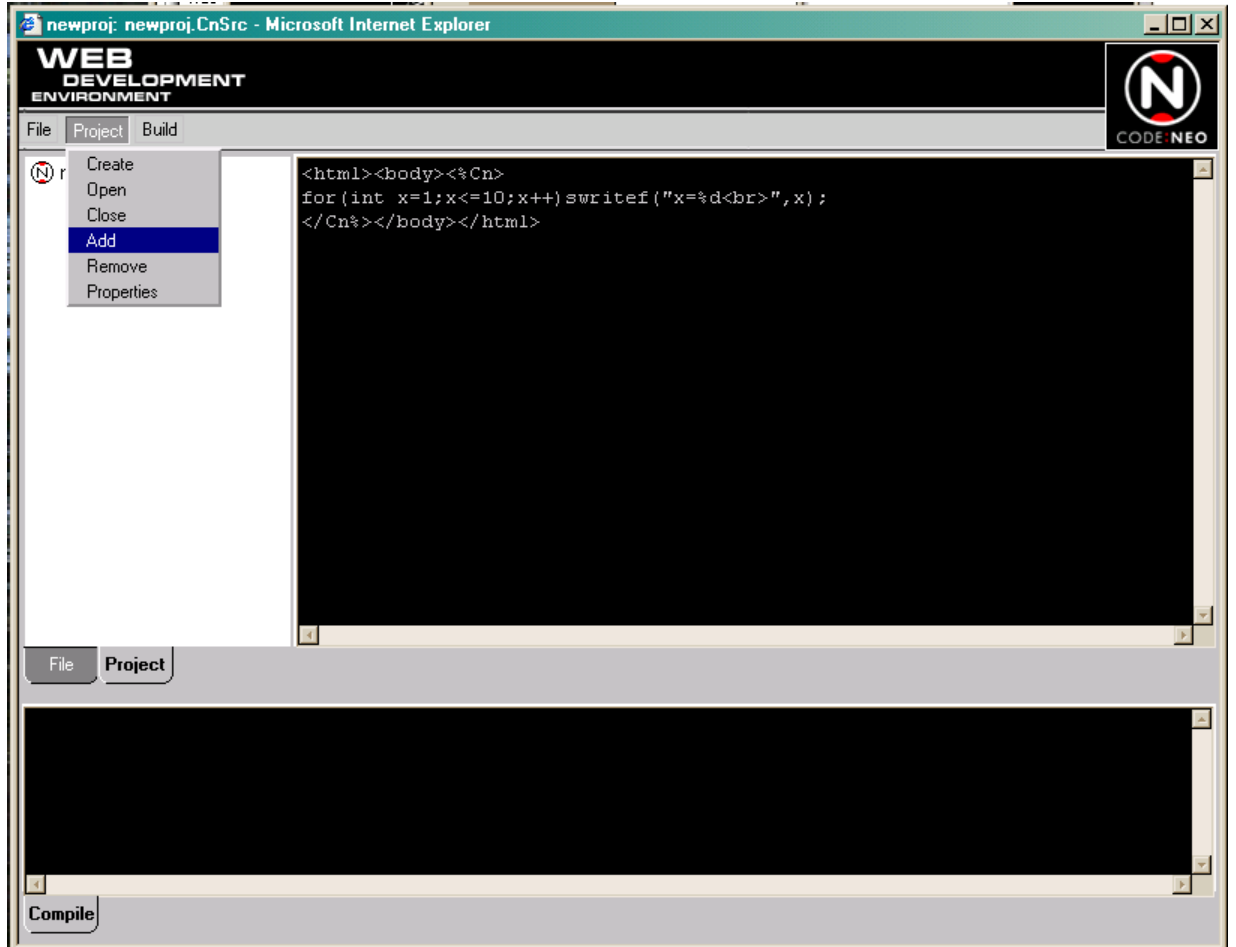
6. type in newproject soucecode and then go to file save, then set filename and click save

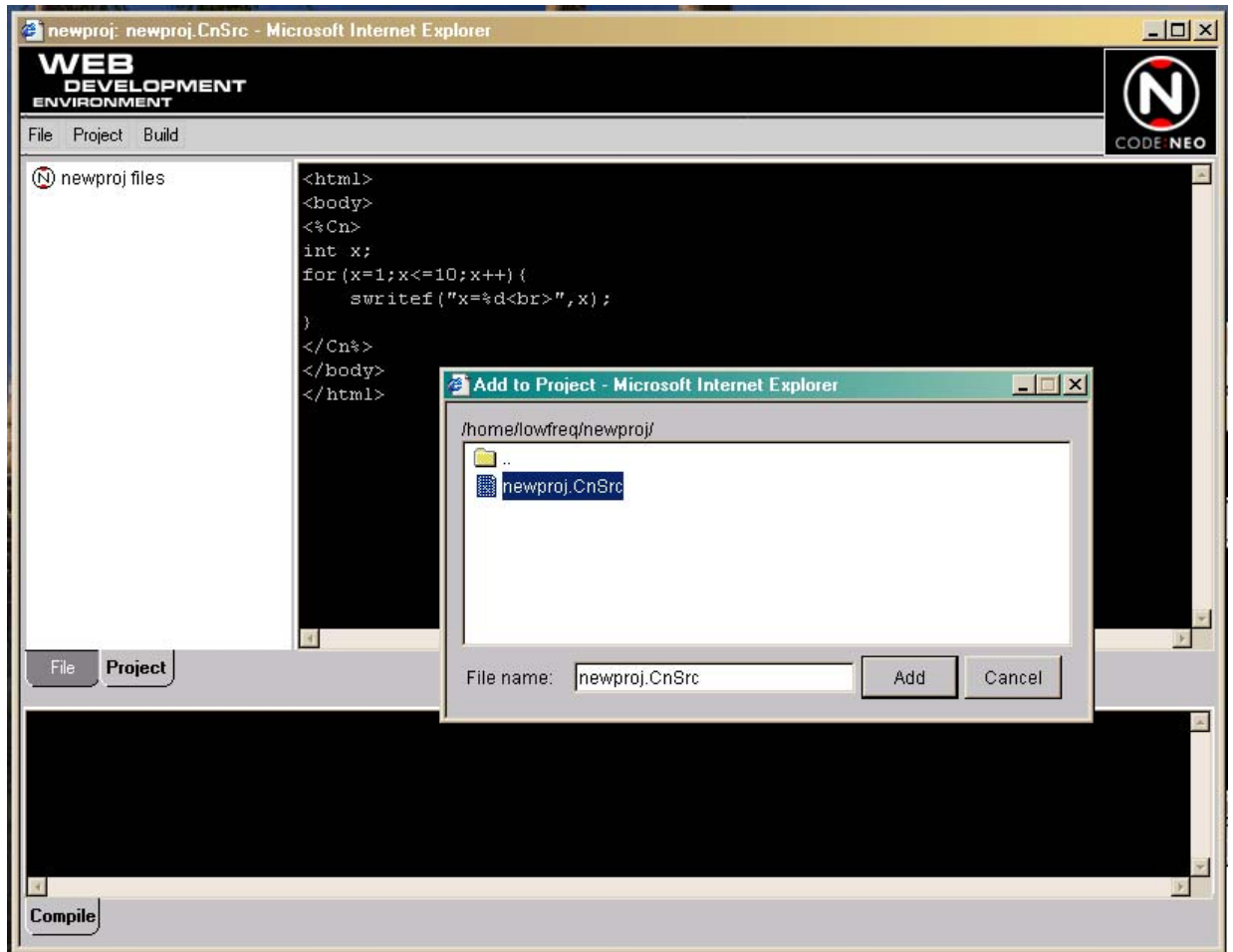


```
<html><body><%Cn%
for(int x=1;x<=10;x++)
    swritef("x=%d<br>",x);
</Cn%></body></html>
```

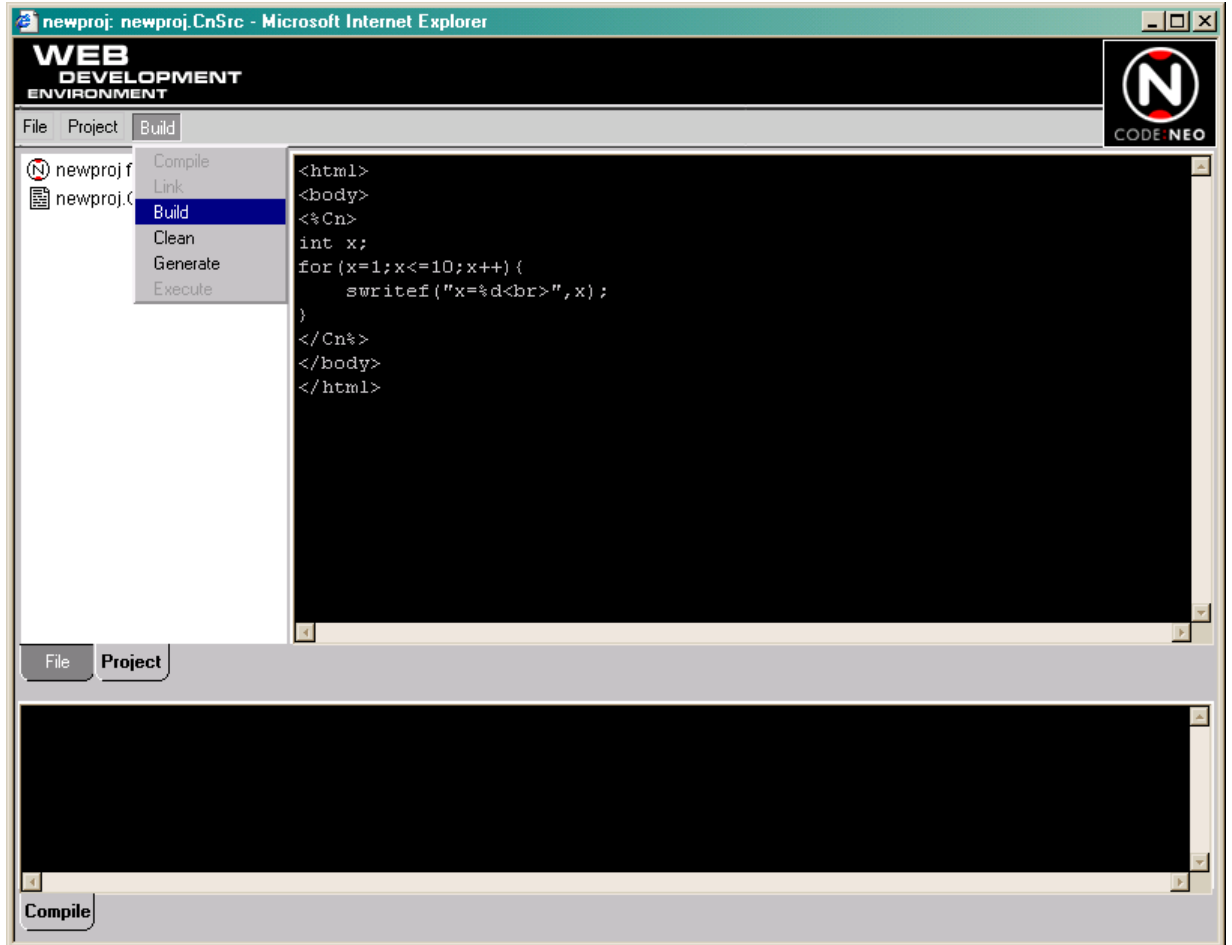


7. add source file to project

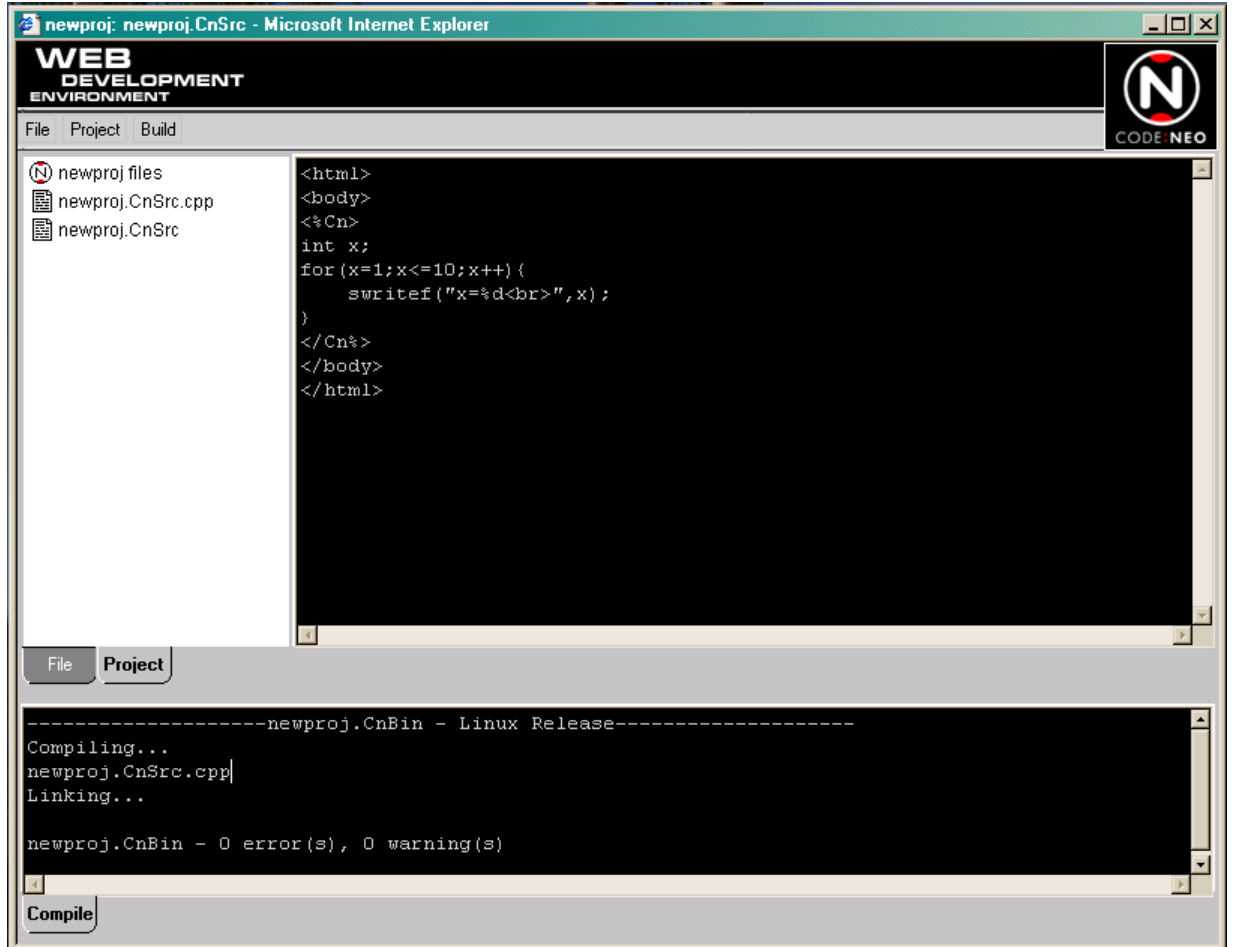




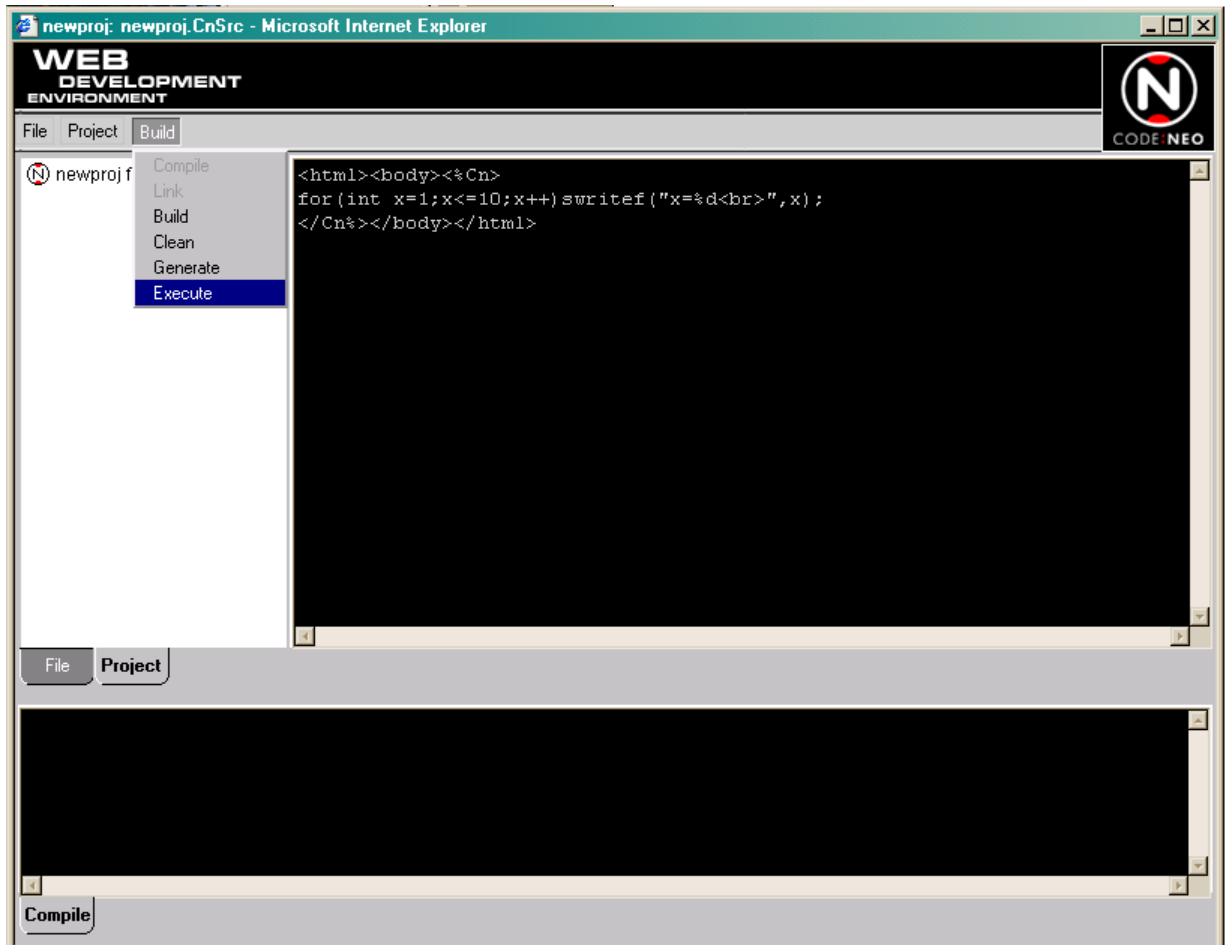
8. build the new project



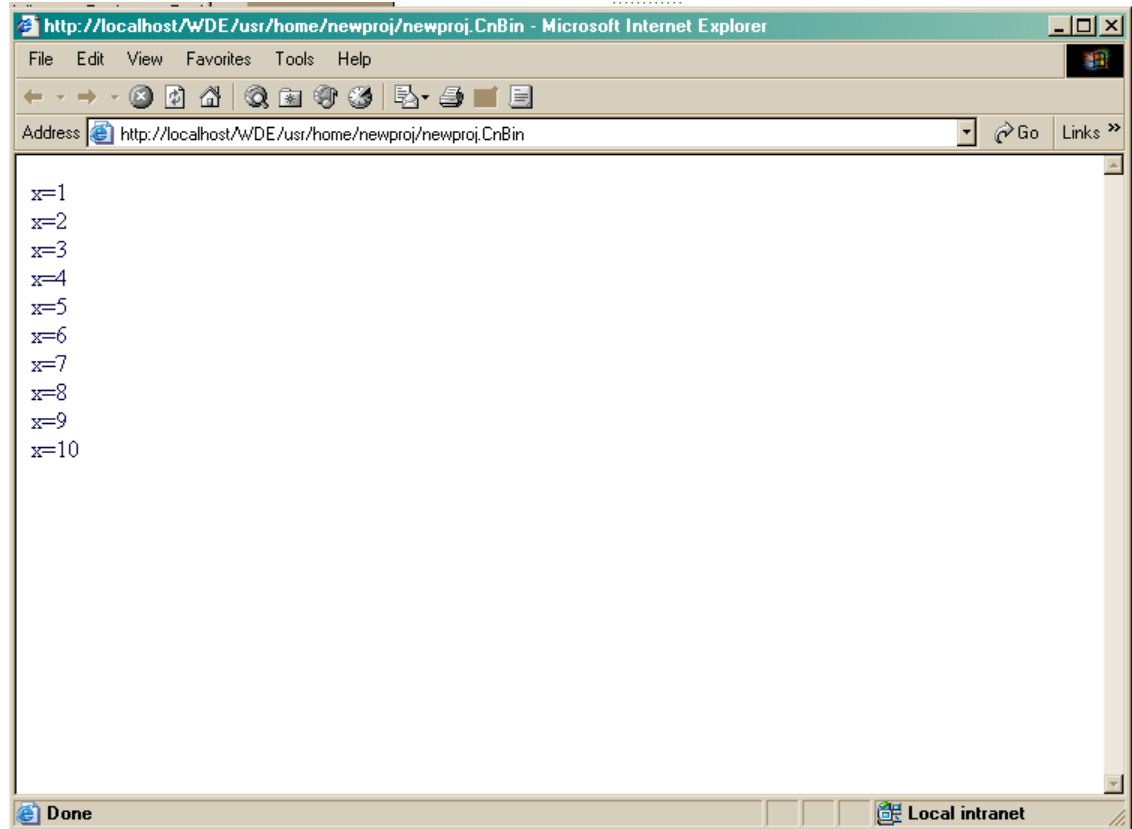
9. check the build results




10. execute the now compiled rivet

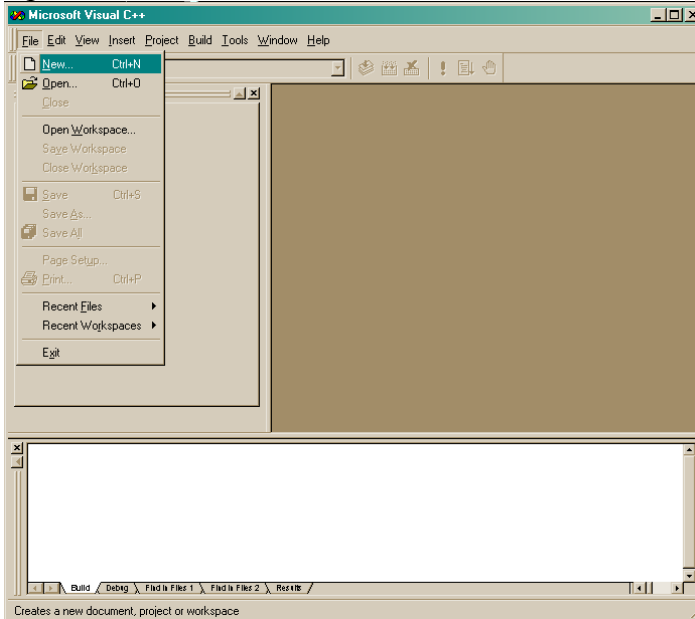


11. see the result of the execution of your first CODE:NEO web application

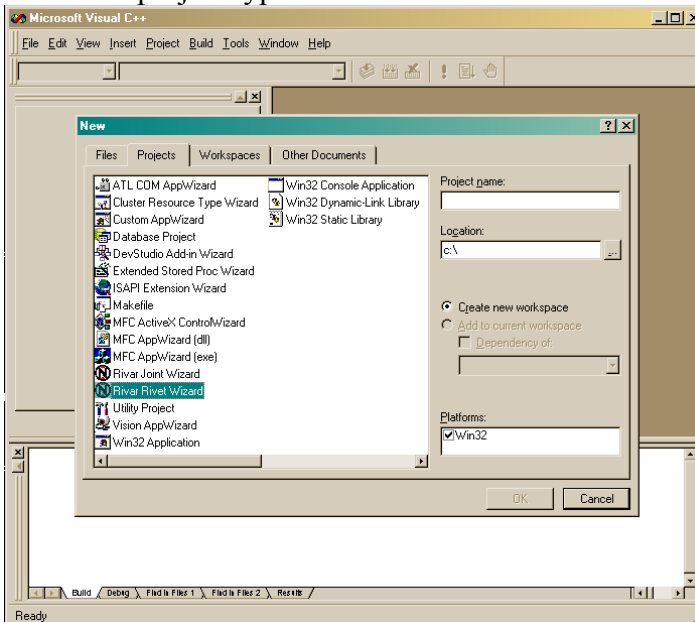


Getting Started With Visual Studio

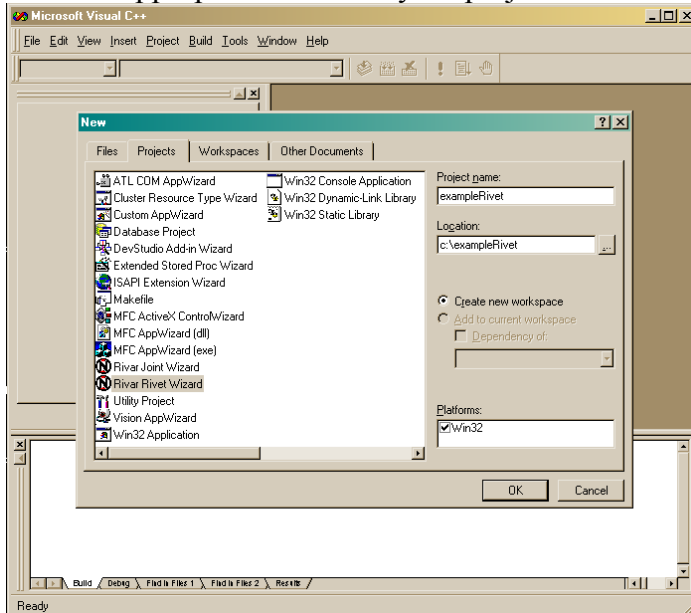
1. Open Visual Studio by clicking on its icon 
2. Open the file menu and click new



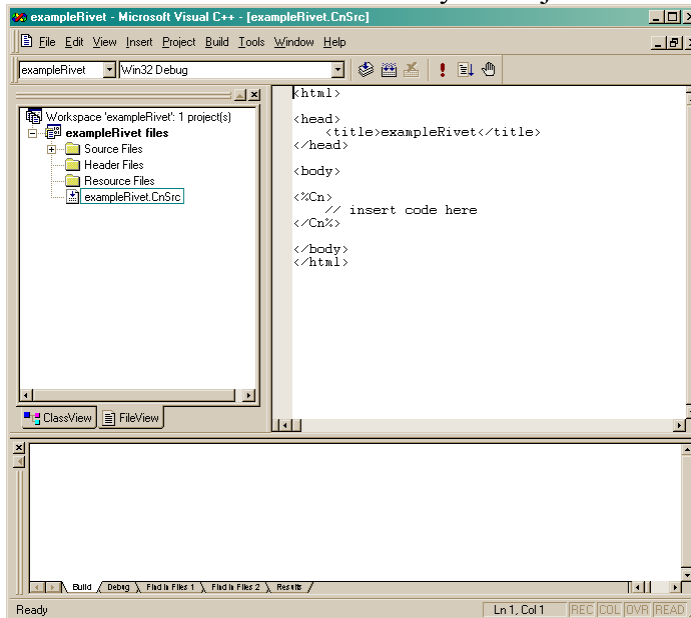
3. Select the project type CODE:NEO Rivet



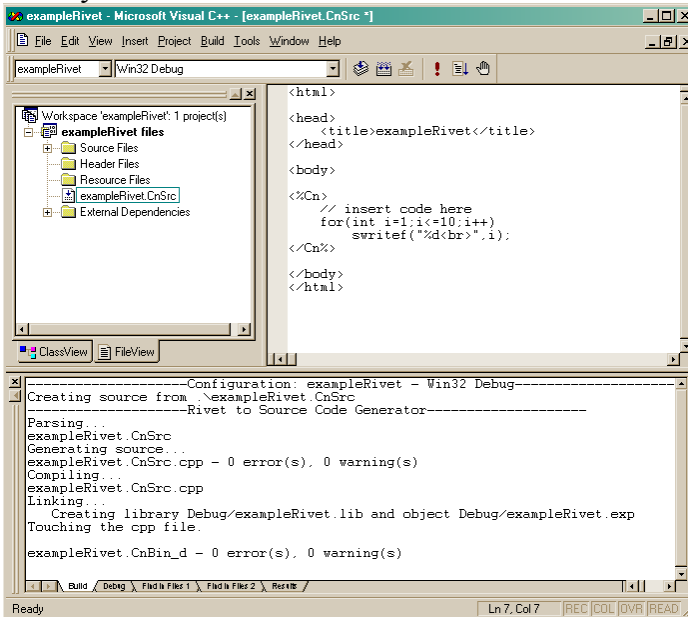
4. Enter an appropriate name for your project



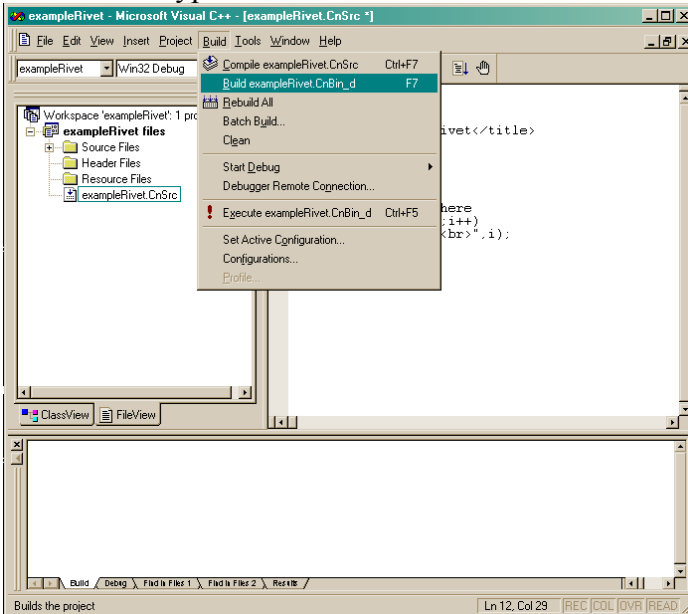
5. Click FileView then double click yourProjectName.CnSrc



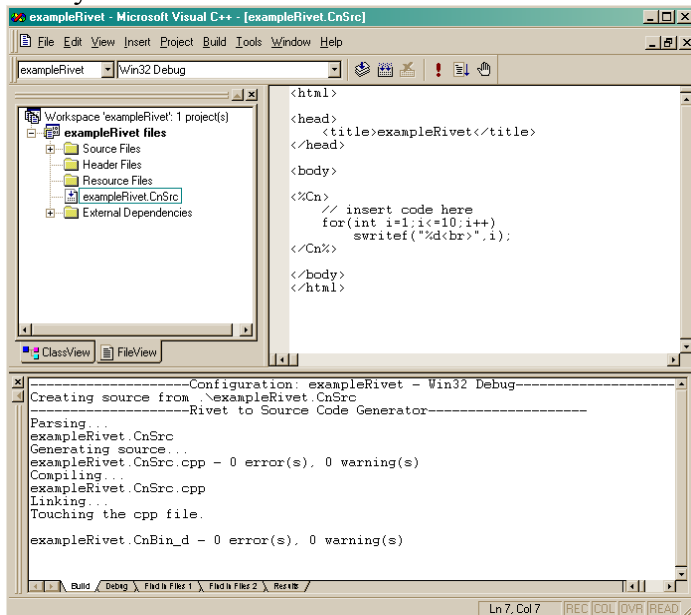
6. Enter your code



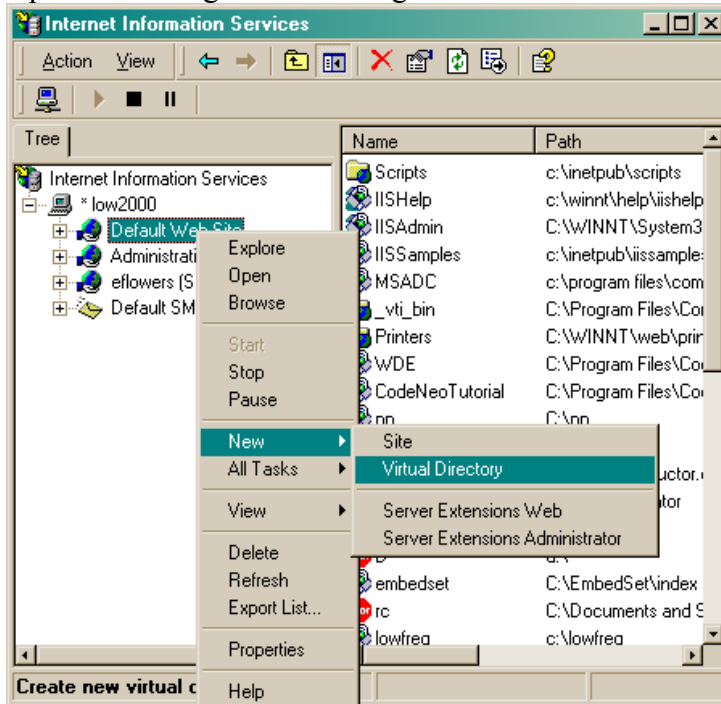
7. Check build type then click build in build menu



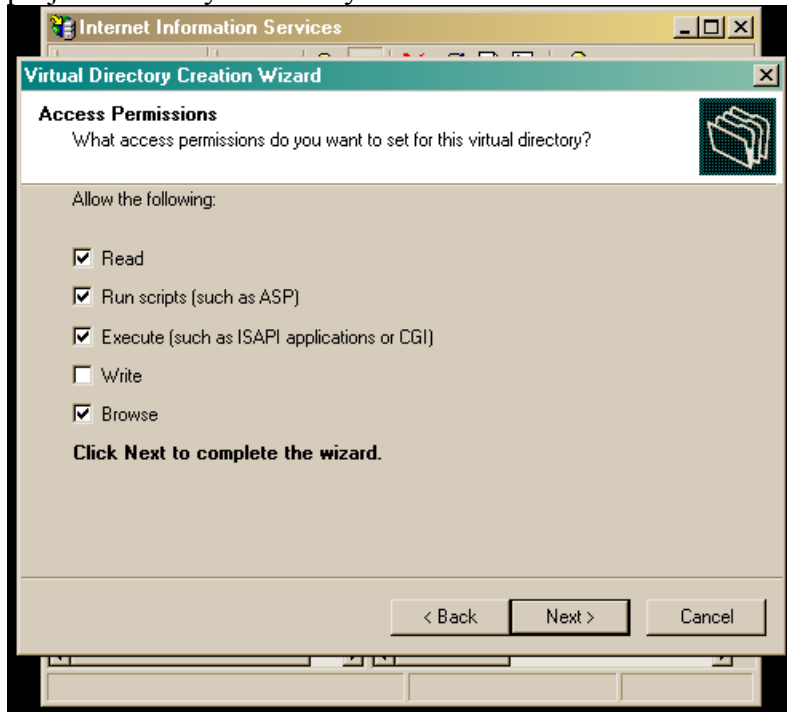
8. Verify Successful Build



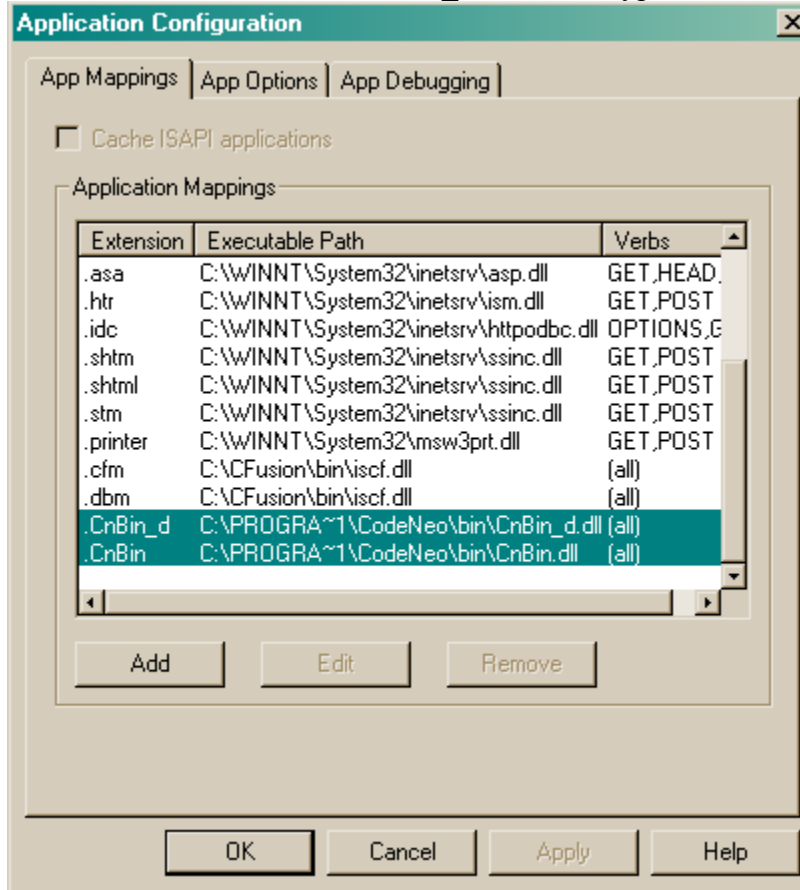
9. Open IIS Configuration and right click on web site and click new virtual directory



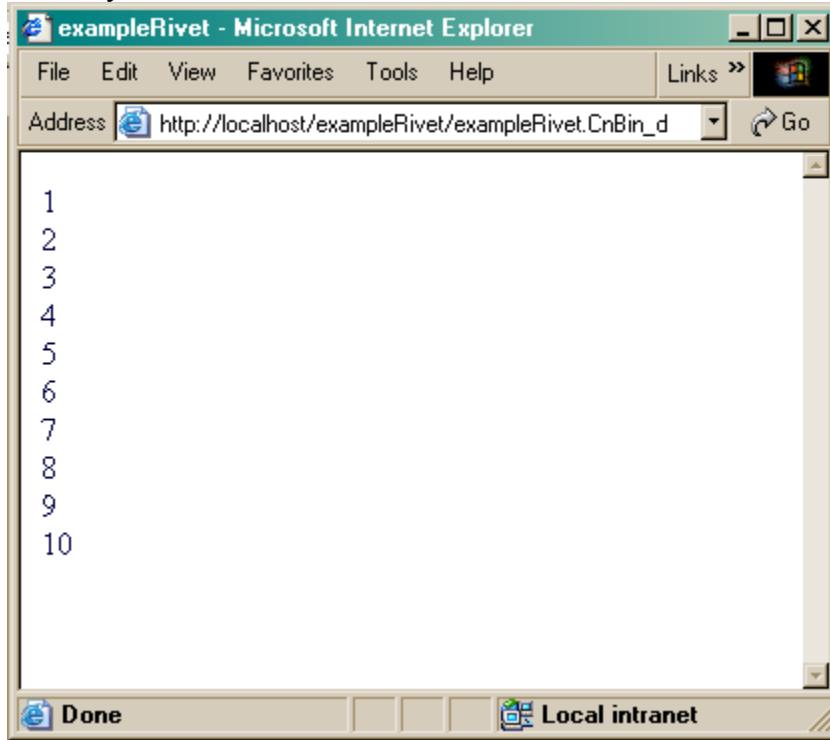
10. Create a new virtual directory, make sure to choose execute, choose browse to make your time of finding it easier, be sure to point this virtual directory at the project directory created by visual studio



11. Make sure the .CnBin and .CnBin_d extension types are enabled



12. Execute your application by pointing your new web browser at your new virtual directory



Debugging CODE:NEO applications;

Several options exist to debug your CODE:NEO application, each has its advantages and disadvantages.

Debugging with trace window

Advantages

- Simple web accessible debugging method
- Only has performance impact in debug version
- Allows rapid debugging
- Takes several format strings for simple programming

Disadvantages

- May not return results under the most extreme circumstances
- It not real-time

Debugging with real-time debugger

Advantages

- Ability to set breakpoints
- Ability to inspect variables
- Ability to step through code blocks
- Ability to easily follow code flow

Disadvantages

- Requires server running 'in process' mode
- Details only accessible on local debugging machine
- Requires web server to be in single client mode

Debugging with logging

Advantages

- Only has performance impact in debug version
- Log is always available and permanent record of execution
- Simple, rapid debug method

Disadvantages

- Not easily accessible over web

Debugging Basics

When you compile any code with your C++ compiler, you have the opportunity to choose what type of compilation that you wish. If you select a debug build additional information is included in the output file so that software can help the programmer to debug the application.

In addition to including extra debug information the CODE:NEO application server treats debug version differently than release versions.

CODE:NEO applications with debug support end in `_d` for example `helloWorld.CnBin_d` is the hello world joint with debug support, whereas `helloWorld.CnBin` is the release version.

ON WINDOWS

Debug versions must be executed by the debug version of the ISAPI extension for IIS. `CnBin_d.dll` instead of `CnBin.dll`.

ON UNIX

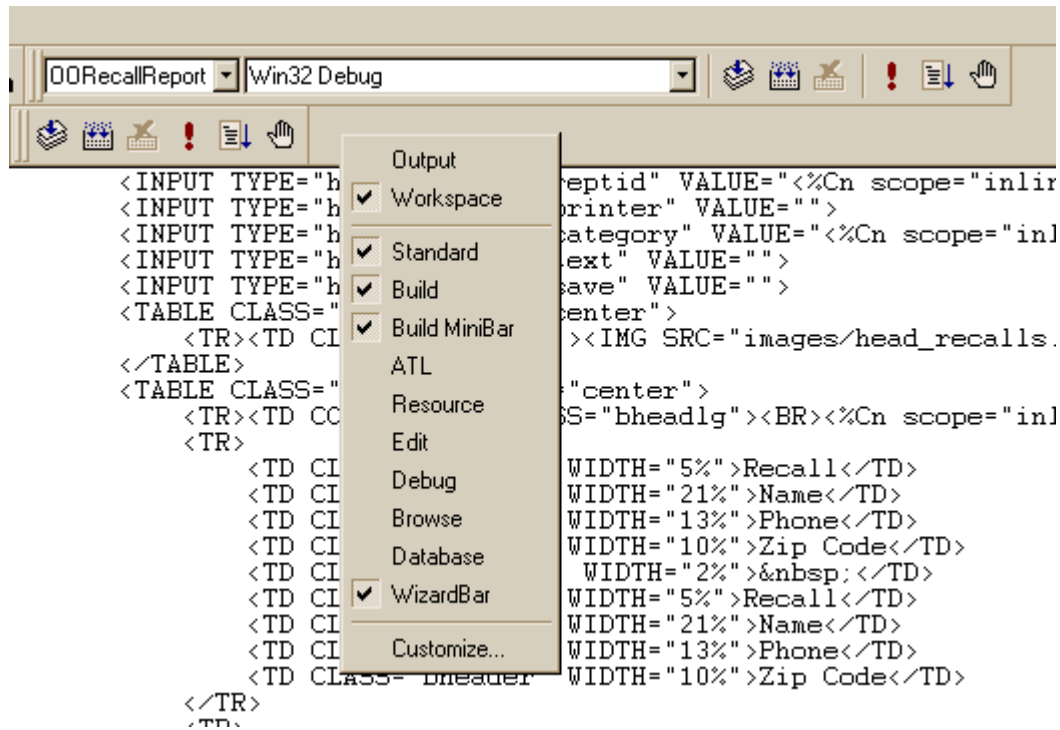
It is possible to run apache web server in a way that it does not detach its TTY. You may do this by specifying `-x` on the command line.

You may then `printf()` within your CODE:NEO Rivet or Joint and see the output on your TTY. This is a fast but rudimentary way of doing debugging.

Choosing the compilation method to include debug support

On Windows

Using visual studio the build mode may be handled by choosing it in the build toolbar accessible by right clicking on a blank area of the task bar menu and choosing the 'build' option.



On Unix

In UNIX the build mode is specified by adding the argument `BUILD=debug` to the make command. For example;

```
make BUILD=debug
```

Will compile and produce a `.CnBin_d` including `CNDEBUG` macro calls.

Real time debugging

Real time debugging with GDB

During the course of programming and debugging our CODE:NEO applications on UNIX variants we use GDB the GNU Debugger.

Log in and establish a security level where you can GDB the apache webserver process. For some systems it's root.

To start gdb, invoke gdb on the executable file. ;

```
% gdb /usr/local/apache/bin/httpd
```

Execute the apache process, telling it to run in 'single' server mode and in the foreground, i.e. do not detach from TTY.

```
(gdb) r -X
```

You may set a breakpoint by;

```
(gdb) break yourfunc
```

You may list your program;

```
(gdb) break yourfunc
```

Please consult the appropriate man pages and web resources for further, more complete information about debugging with GDB.

Real time debugging with kGDB

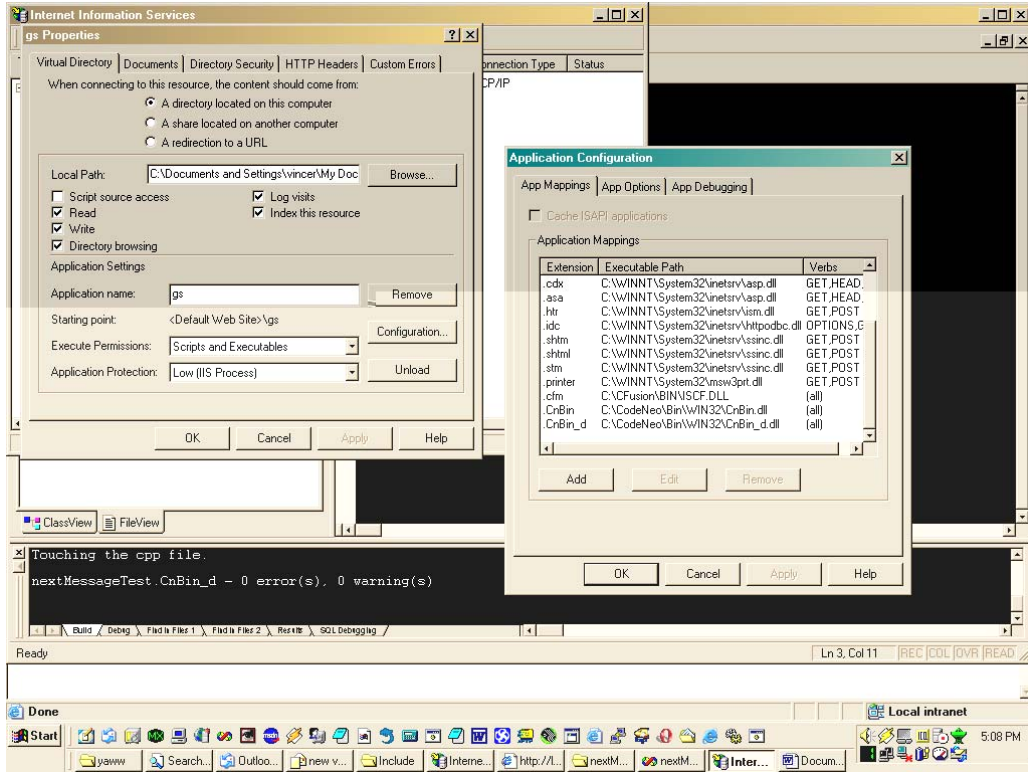
The steps for debugging with kGDB are similar for GDB, only you will perform the operations in a more graphical manner. Load the apache daemon as the program to debug and call it with the **-X** argument so that it does not detach from the TTY and executes only one web request at a time.

We find that this is the most productive way to debug CODE:NEO applications as kGDB allows fast easy access to real-time debugging on UNIX variants.

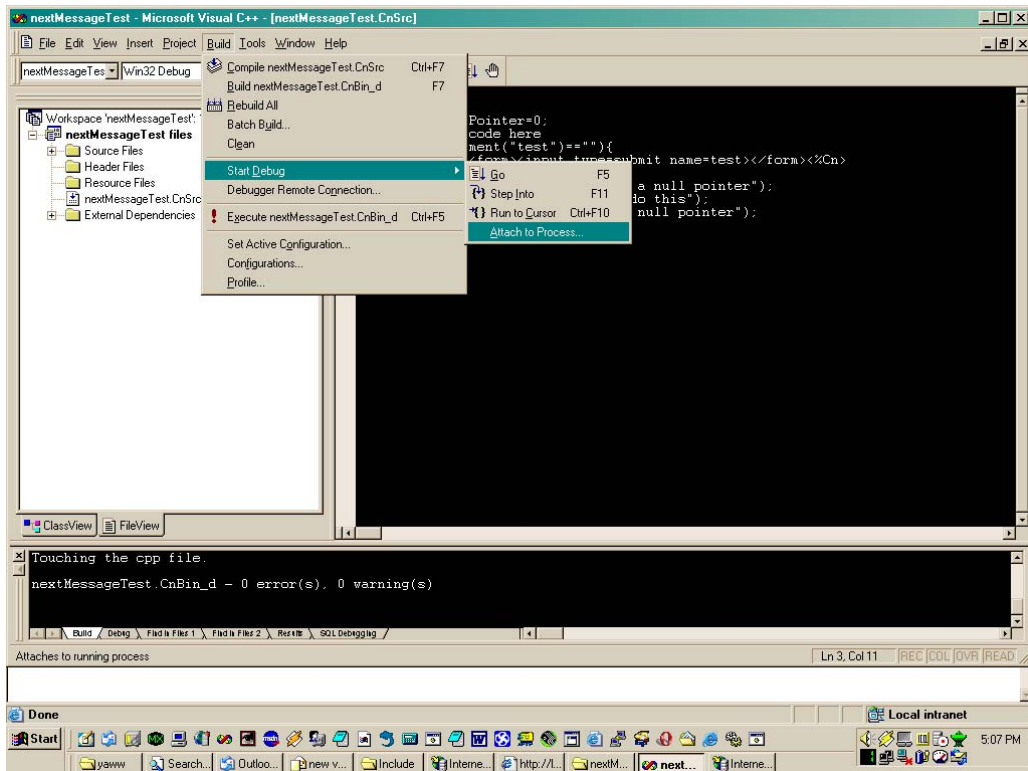
Real time debugging with Microsoft Visual Debugger

If you are on the windows platform you may use the Microsoft Visual Debugger, which is included with Microsoft Visual C++ 6.0. It will allow you to step through and analyze your CODE:NEO application just as you would any C++ application.

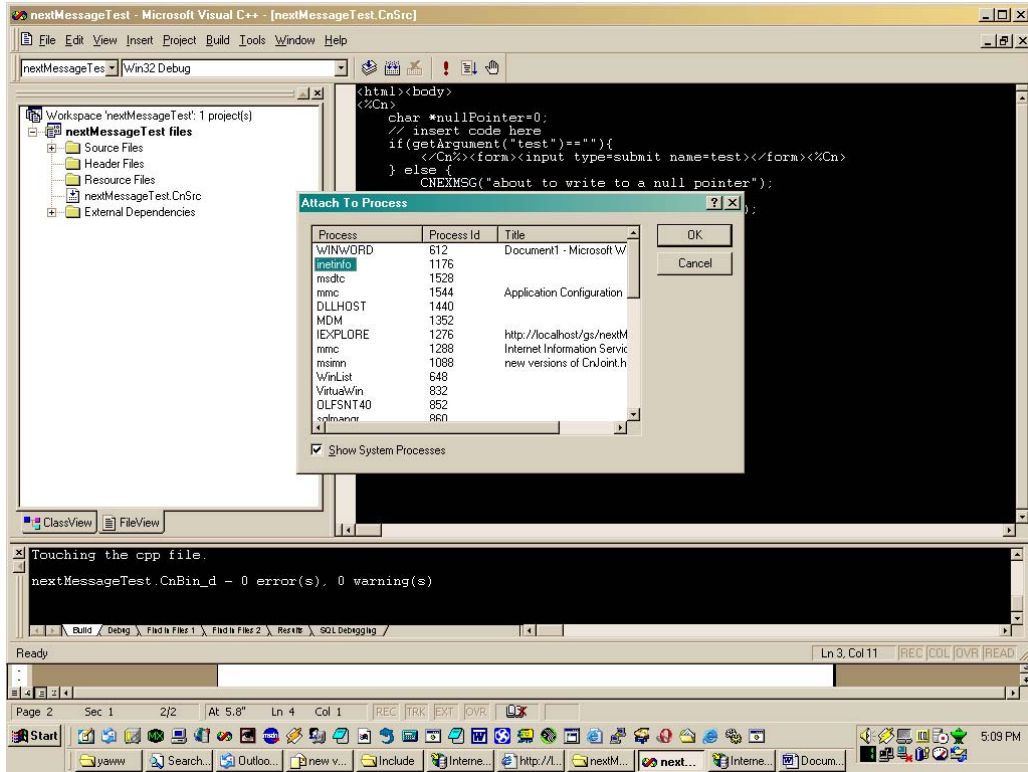
Verify the iis web settings;



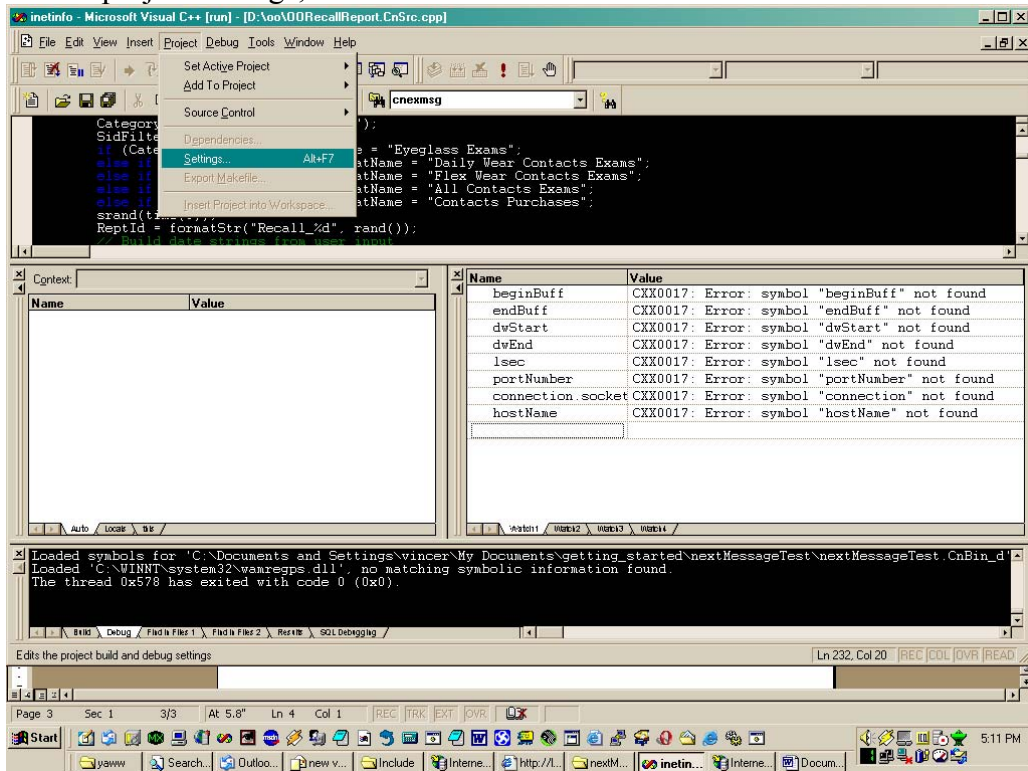
Attach the debugger;



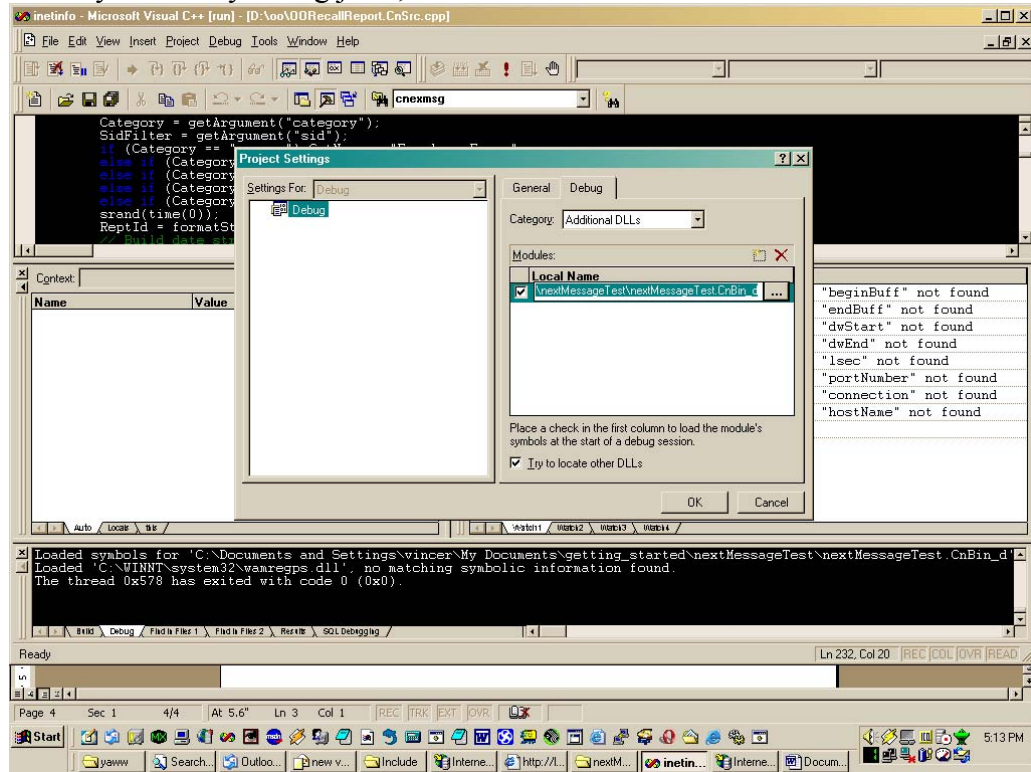
Click show system process and choose inetinfo.exe;



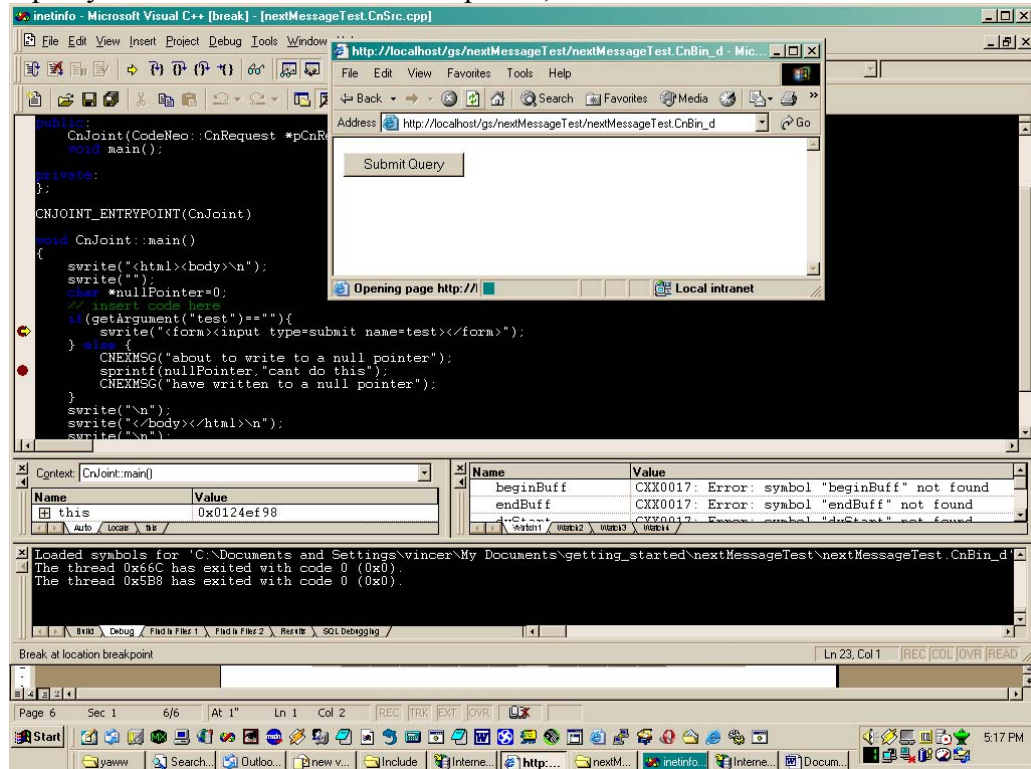
Set the project settings;



Select your binary debug joint;



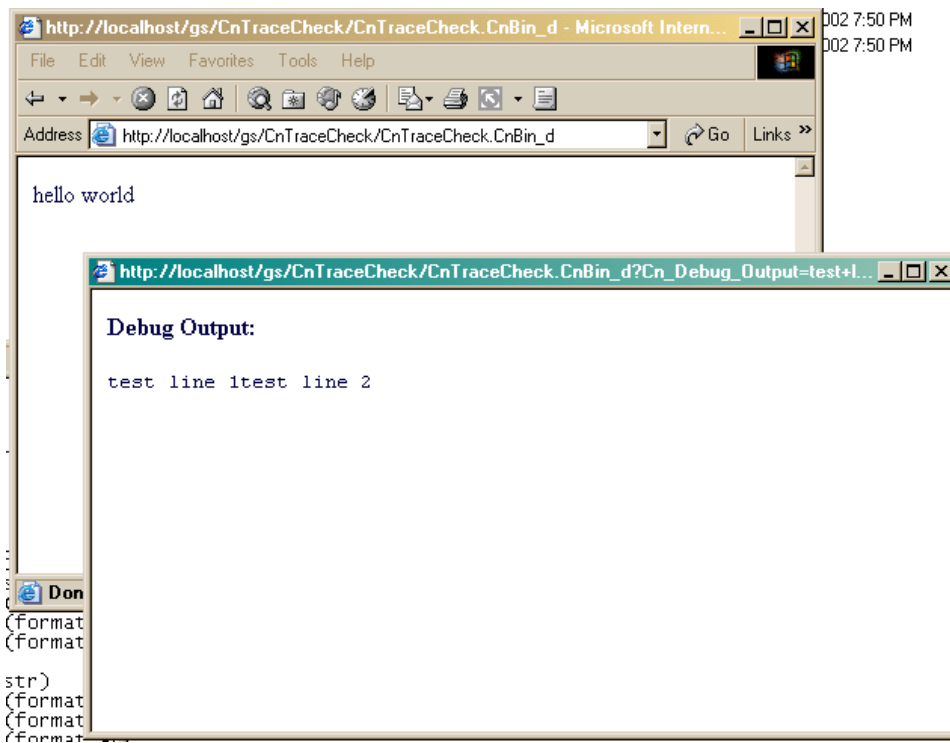
Open your source file and set breakpoints;



the yellow cursor will mark where your point of execution is. You may then continue to use your debugging controls as normal to step through or analyze your code.

Real time debugging with CODE:NEO debug trace window

CODE:NEO provides functionality for opening a trace window that contains line of detail that the programmer adds for debugging. This window will optionally appear when the project is compiled in debug mode, creating a .CnBin_d file.

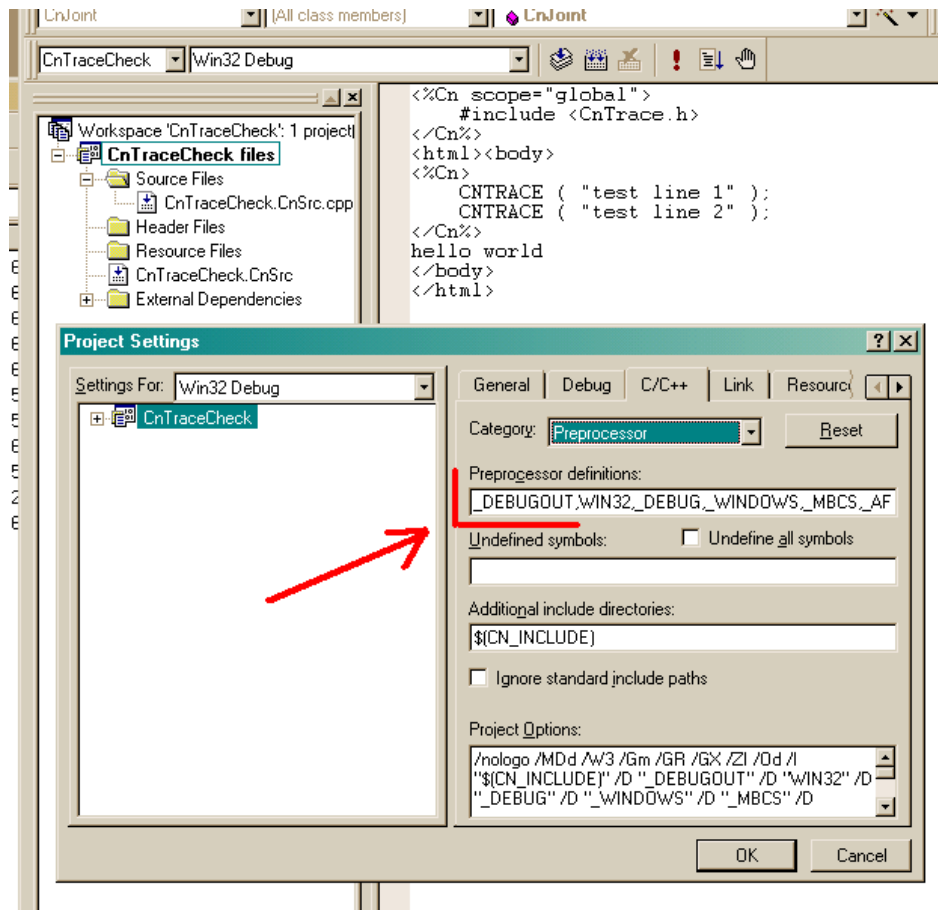


is produce by the Rivet code;

```
<%Cn scope="global">
    #include <CnTrace.h>
</Cn%>
<html><body>
<%Cn>
    CNTRACE ( "test line 1" );
    CNTRACE ( "test line 2" );
</Cn%>
hello world
</body>
</html>
```

and including the preprocessor directive, “_DEBUGOUT”, without the quotes.

On Windows



or;

On Unix

edit Makefile, to include `"-d_DEBUGOUT"` to the compiler options.

Debug Inclusion

CODE:NEO includes a macro called `CNDEBUG`, which is capable of including debug code only when the project is compiled in `DEBUG` mode. That is to say that any commands, which exist within the `CNDEBUG` macro, are present in the debug build but not the release build.

```
CNDEBUG ( command; )
```

Command will be present in the .CnBin_d but not .CnBin.

Common use might be;

```
<%Cn scope="global">
    FILE *LOGFILE;
    void OpenLog(){
        LOGFILE=fopen("c:\\ooRecallLog.txt","a");
        time_t ltime;
        time( &ltime );
        fprintf(LOGFILE,"\nNOORECALL REPORT: %s\n",ctime(&ltime));
    }
</Cn%>
<%Cn>
    CNDEBUG ( OpenLog(); )
    if (SQLExecDirect(SqlHnd, (unsigned char*)SqlStr.c_str(),
        SqlStr.length())!=SQL_SUCCESS){
        Errors += "Failed to query EXAMCODE table<BR>\n";
        GetSqlErr(Errors, SqlHnd);
        CNDEBUG ( fprintf(LOGFILE,"ERROR: SqlHnd SQLExecDirect\n"); )
    } else {
        CNDEBUG( fprintf(LOGFILE,"SUCESS: SqlHnd SQLExecDirect\n"); )
    }
    CNDEBUG ( fclose(LOGFILE); )
</Cn%>
```

This would allow optional logging based on whether the author was building in debug or release mode.

Debug Using Logging

You may use the CNDEBUG macro to include code during the debug process that logs output to a file. The file gives you a permanent record of the execution that did occur.

To do this you may either build your own logging routines or use our logging service built into the CnUtil library.

Using Cookies With CODE:NEO

A cookie can be used with or without the shared memory manager to maintain state with the web browser client. That is to say individual parameters can be stored so that the application may re-associate the http connection with a given session of activity. By storing the information for a session in the shared memory manager by using cookies along with CnSession CODE:NEO applications can execute very fast!

You may retrieve the value of a given cookie with the `getCookiesValue()` method of a `CnHttpRequest`. To access that member from a joint or rivet, use the joints, `pCnHttpRequest`.

```
if (pCnHttpRequest->getCookiesValue("cookie") !="" ) {
    // process that cookie
} else {
    // most likely go about setting the cookie
}
```

You may use the `setCookie()` method to set values for cookies.

```
setCookie("cookie","something");
```

You may also specify the path and expiration date such as;

```
setCookie("name","value","path","Wed 10-Oct-2001
08:00:00 GMT");
```

Setting the expiration date to sometime in the past makes a memory resident cookie that goes away after the browser is closed.

Using `sprintf` and other `CnJoint` functions outside the `CnJoint` object.

Normally in the course of CODE:NEO programming you want to create global functions, which provide a service from use within your rivet.

Because global functions are outside the scope of the `joint::main()` they may not call `sprintf()` directly. However often this is a wanted functionality.

Take the following example;

```
<%Cn scope="global">
CnHtmlServer *ptrServer;
char *getFile(char *fname){
    char *outbuff;
    FILE *fptr;
    outbuff=(char *)malloc(5096);
    fptr=fopen(fname,"r");
    memset(outbuff,0,5000);
    if(outbuff!=0){
        while(!feof(fptr)){

fgets(outbuff+strlen(outbuff),5000,fptr);
        }
        fclose(fptr);
        return(outbuff);
    } else {
        strcpy(outbuff,"Error Occured Loading
File");
        return(outbuff);
    }
}
void incFile(char *fname){
    char *incBuff;
    incBuff=getFile(fname);
    ptrServer->sprintf("%s",incfile);
    free(incBuff);
}
</Cn%>
<html>
<body>
<%Cn>
    // set the ptrServer so that it can be used
globaly
    ptrServer=this;
    // write to the response
    incFile("/export/home/store/htdocs/headertemplate.
html");
</Cn%>
</body>
</html>
```

This code allows you to include a file into the output of the rivet from a global function. This is because we *export* **ptrServer** for use within the global function **incFile()**.

This technique can be used to utilize any of the objects available such as **CnHttpRequest**, **CnHttpResponse**, Etc....



The CODE:NEO application server is a web server add-on for Apache, IIS or other ISAPI / module compliant web server a shared memory server and set of libraries for executing CODE:NEO applications.

The Development Tool is software which assists a developer in creating a CODE:NEO application. These tools include a generator to turn pages which include HTML and C++ into a specified format C++ file for compilation. These tools include the header files and API documentation as well as tutorials and examples to get the programmer started.

The Application is actually the end result product of the development tool. This is the binary code that will actually be executed by the server's processor natively instead of abstractly as is true of JAVA, ASP, .NET, J2EE, Etc. The benefits over these technologies are speed and power. As CODE:NEO allows programming to be done in C++ you do not have the limits of web languages like ASP, .NET and JAVA. From within a web page you have the power of C++. The application result files are assembled binary protecting the developer's intellectual property rights.