Marten - CGI Example

This document will show you how to create a CGI executable using the Marten and Xcode development environments.

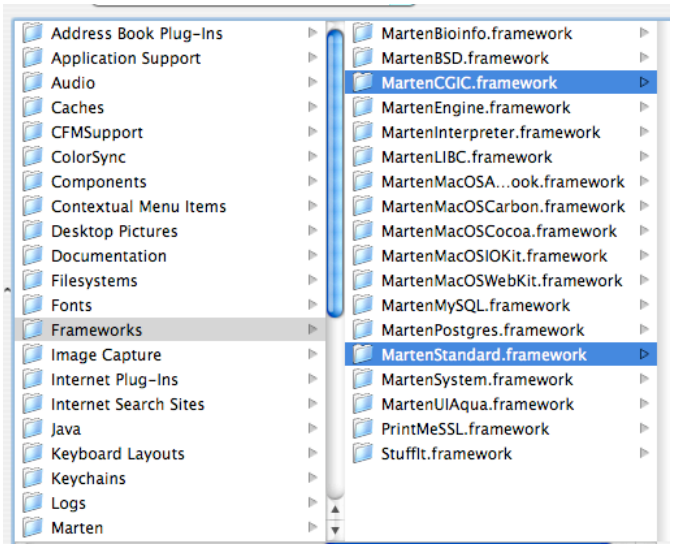
Important - Read Me

A CGI executable is a command line tool and as such cannot be created by using the Marten Update command. And because the CGIC library supplies the "main" method, the Marten Build command cannot be used either. In this case, the exported C files of a CGI executable must be compiled and linked by a third party C language development environment. In this example, the Apple Xcode application is used as that development environment and users should have the Xcode environment installed on their system. Because these files will be linked against Marten frameworks, these frameworks must have been installed in the appropriate location for users to complete this example.

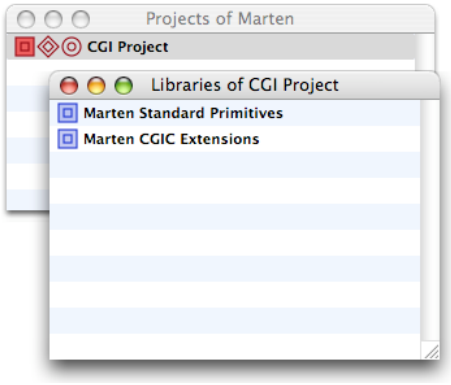
This example requires special Marten C code files. These may be obtained from the support page of the Andescotia website at "http://www.andescotia.com/support/" by downloading the file called "CGI Example Files". The C files will be added to your soon-to-be-created project folder.

Creating the Marten project

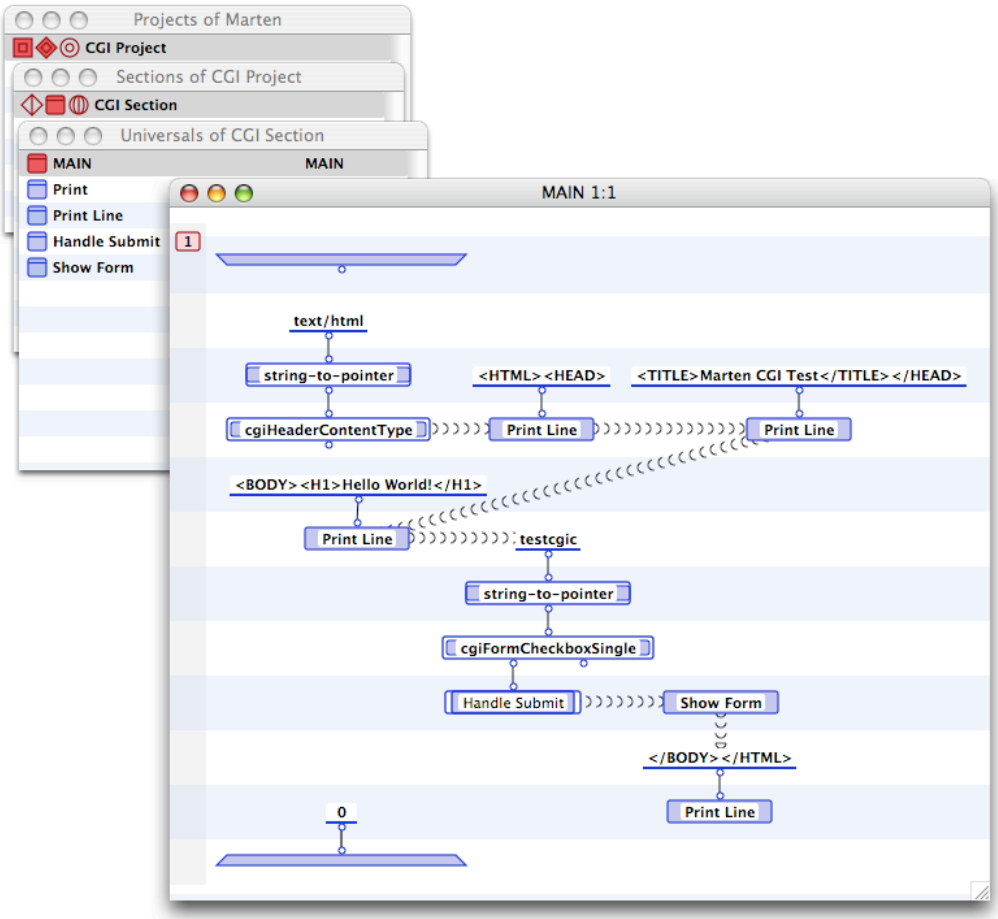
To begin, launch the Marten IDE and create a new Marten project in a folder called "Marten CGI Example". Name the project and project application "CGI Project" and add the CGIC and Standard libraries to the project. These libraries are MacOS X frameworks and should be found in one of the standard frameworks folders:



The libraries can be displayed in the Project Libraries window:

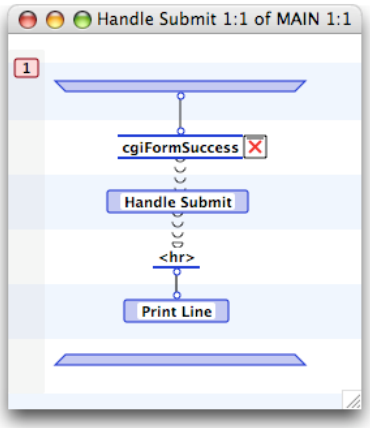


Create a new section and name it "CGI Section". Then create 5 universal methods as shown below. The universal method MAIN should be installed as the MAIN method and the code is presented below.

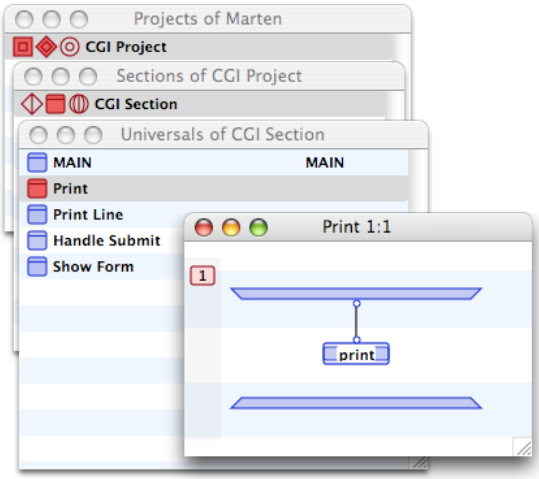


The MAIN method creates the basic HTML for the CGI executable. The browser window title will be "Marten CGI Test" and the text "Hello World!" will be displayed at the top of the page. A "submit" button named "testcgic" is on the form and it is checked for submission in the local "Handle Submit". Finally the universal "Show Form" is called and the HTML is completed.

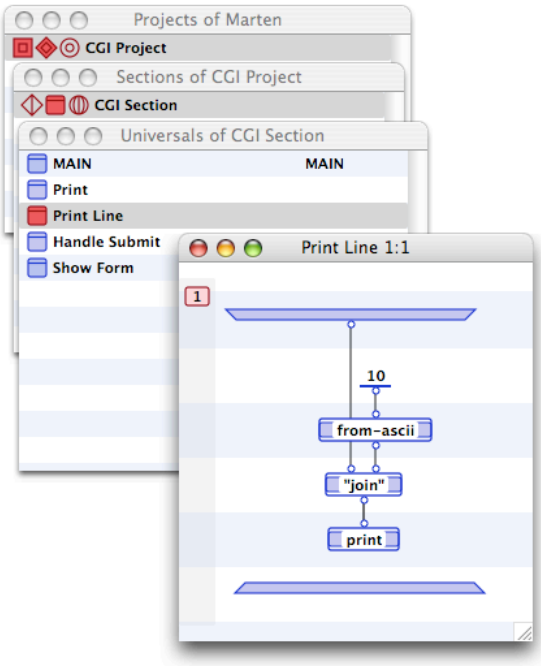
The "Handle Submit" local simply tests for submission (terminates if not submitted) and if successful, calls the "Handle Submit" universal and then finishes with a horizontal rule.



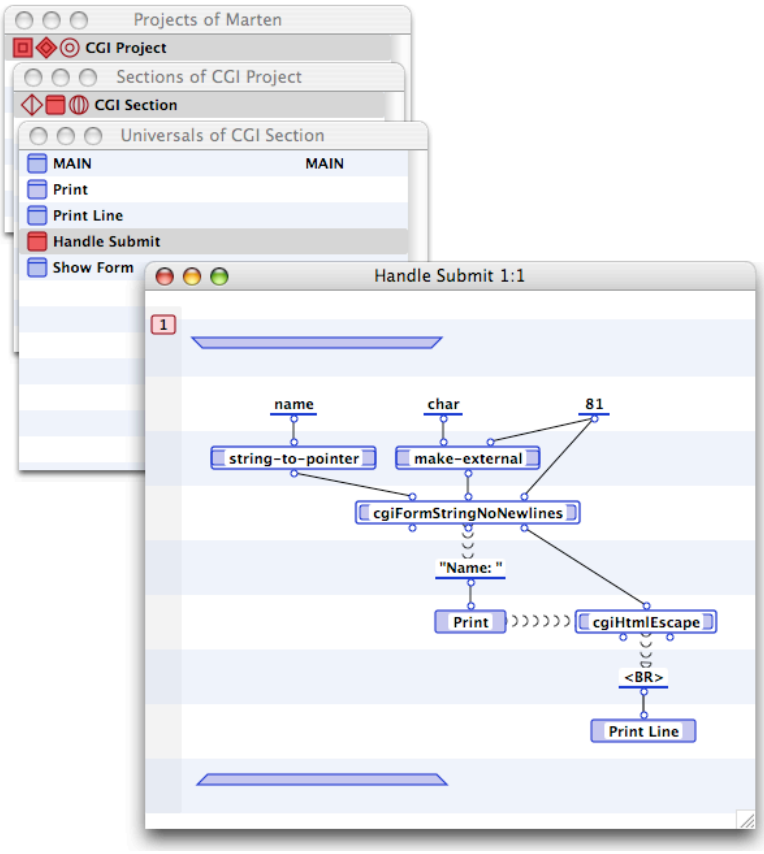
The Print universal is a simple wrapper for the "print" primitive.



Similarly the Print Line universal is a simple wrapper for the "print" primitive combined with a Unix (ASCII 10) carriage return.

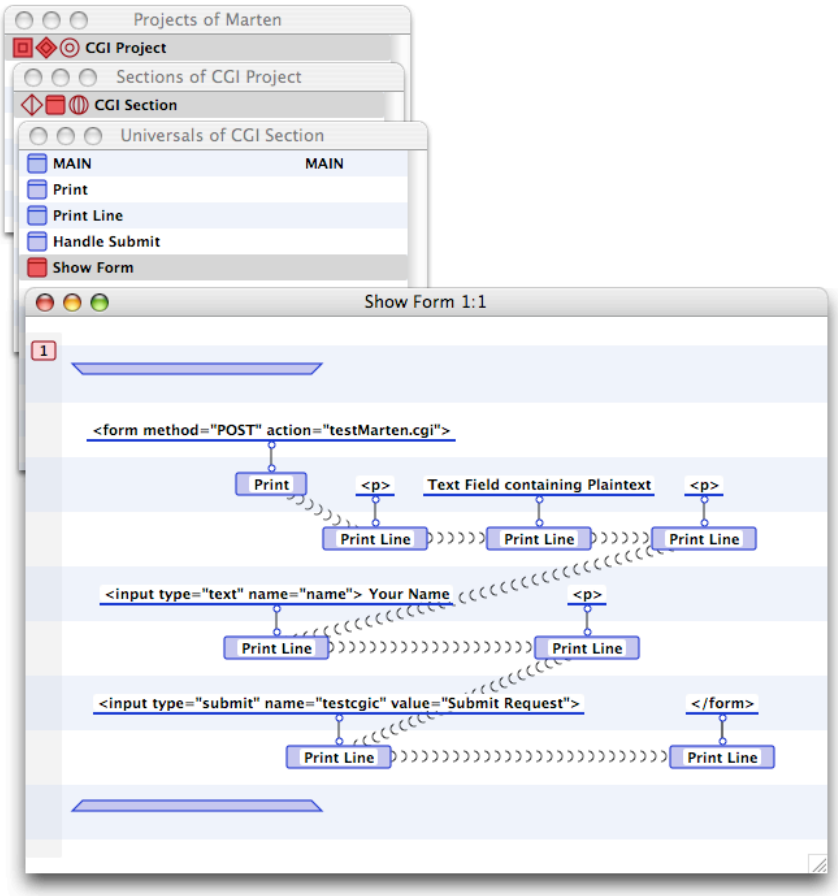


The "Handle Submit" universal is called when the Submit Request button is pushed. It gets the text from the text input box named "name" and prints it after the phrase "Name: ".

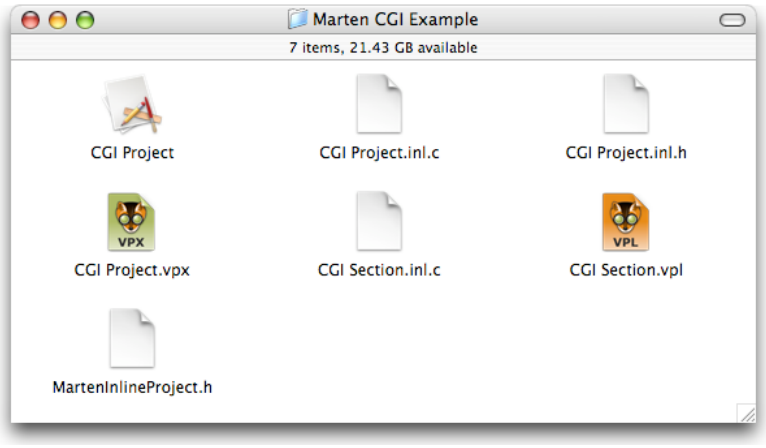


Finally the "Show Form" universal is responsible for the majority of the page. It is a form whose action is to call the CGI executable named "testMarten.cgi", which will be the name of the executable created in this example. The page will display the

text "Text Field containing Plaintext" followed by a text input box labeled "Your Name". The final object displayed will be a submit button with the title "Submit Request".



Save the project and section. Then export (in the inline format) the project and section to the project folder. When finished, the contents of the folder should be:



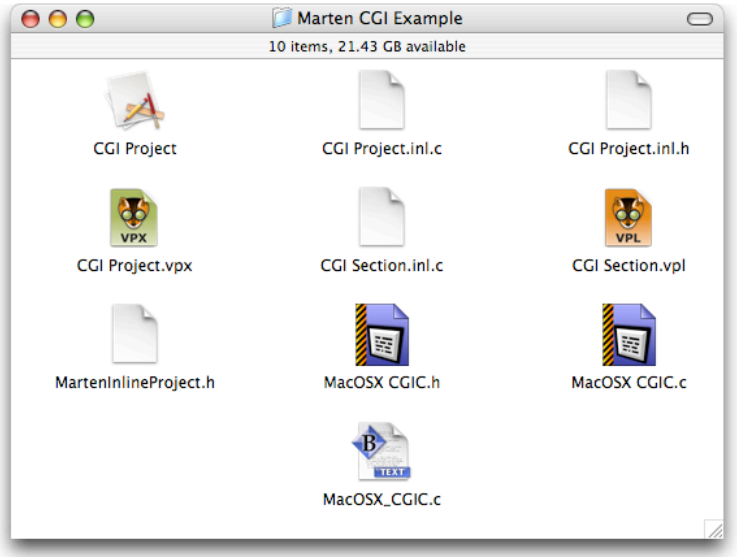
Quit the Marten application.

Obtaining the CGI C Files

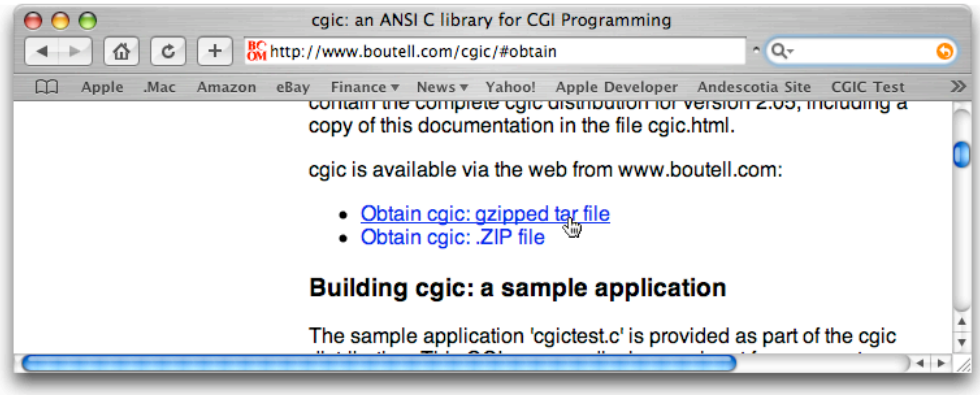
At this point, add the three files obtained from the Andescotia website to the project folder. These files are ""MacOSX CGIC.c",

"MacOSX CGIC.h", and "MacOSX_CGIC.c".

Be sure to hang on to the "MacOSX xxx" files after completing this example as they will be used as the basis for a future discussion of the Marten library file format and as an illustration of how to write Marten primitives, procedures, structures, and constants.

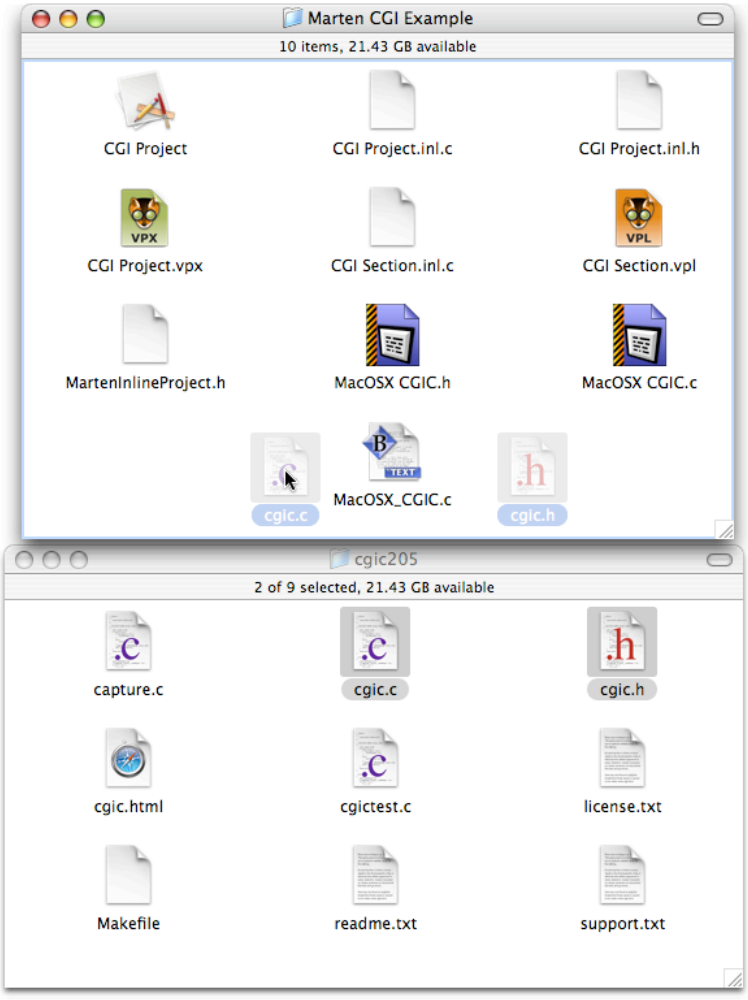


There are still C files that are required for this example. The Marten CGI library is based on the shareware CGI C library created by Thomas Boutell. As Andescotia LLC does not yet have a license to distribute this library, the actual C files required to create an executable must be downloaded from his website. They can be obtained from www.boutell.com as shown below:



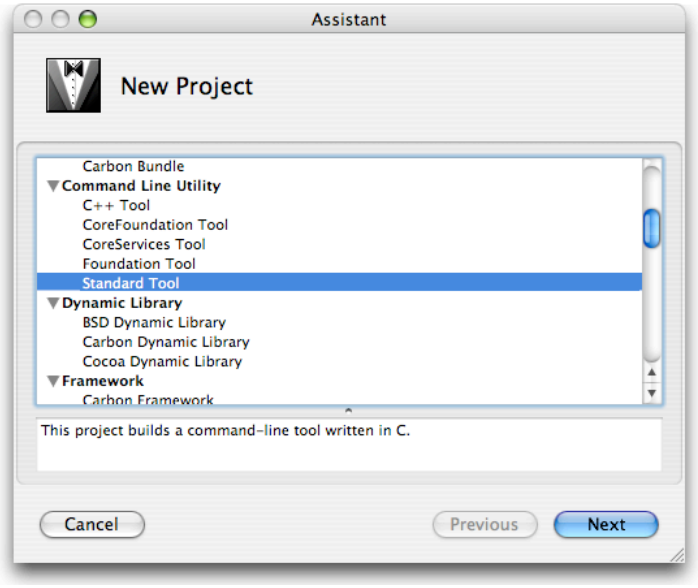
After the gzip file has been downloaded, uncompress it to create the "cgic205" folder.

The files that we need are the cgic.c and cgic.h files. Copy them into the project folder as illustrated below.

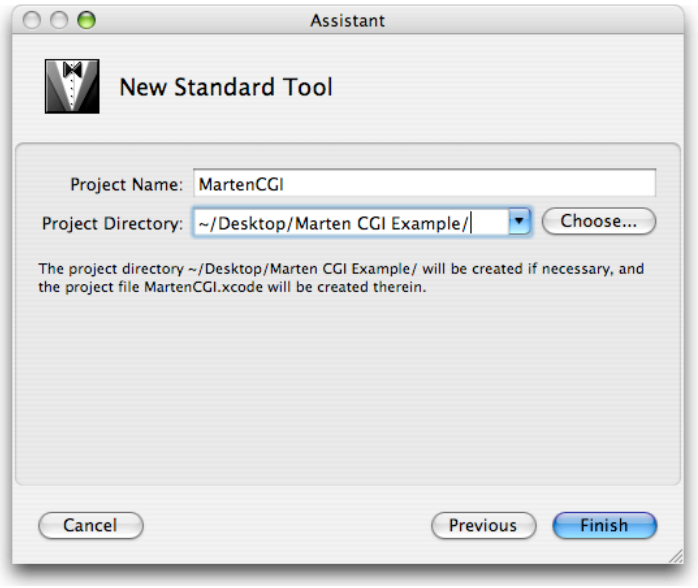


Preparing the Marten Xcode project

Now that all the C files are available, it is time to create the Xcode project. Launch Xcode and create a new project. It should be a "Standard Tool" Command Line Utility.

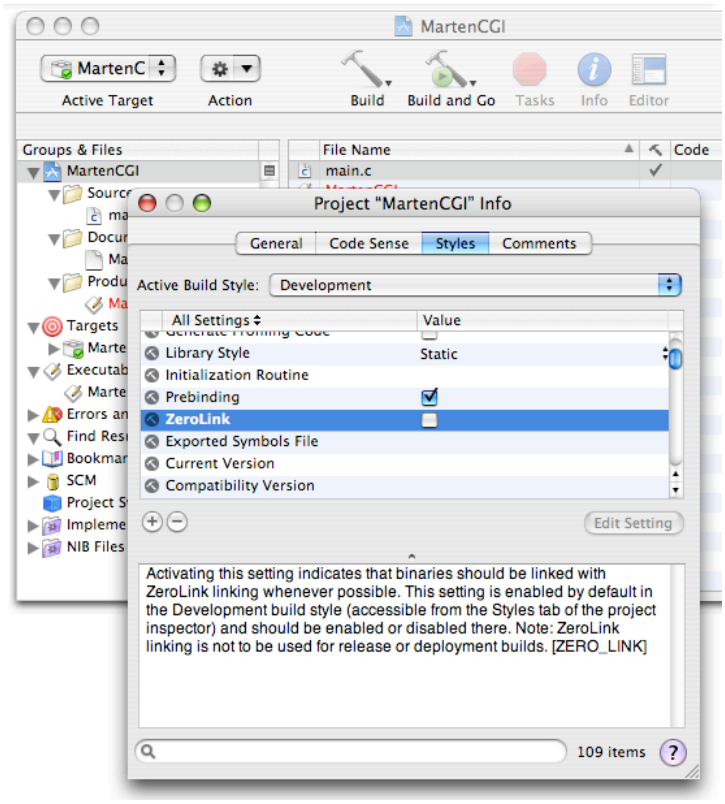


Name the project MartenCGI and point the project directory to the project folder.

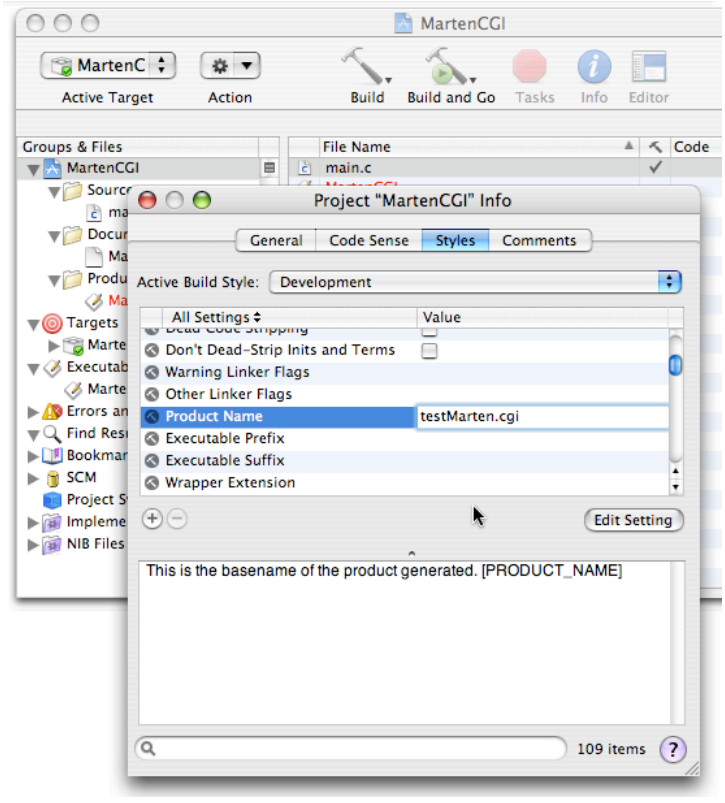


Marten Xcode project settings

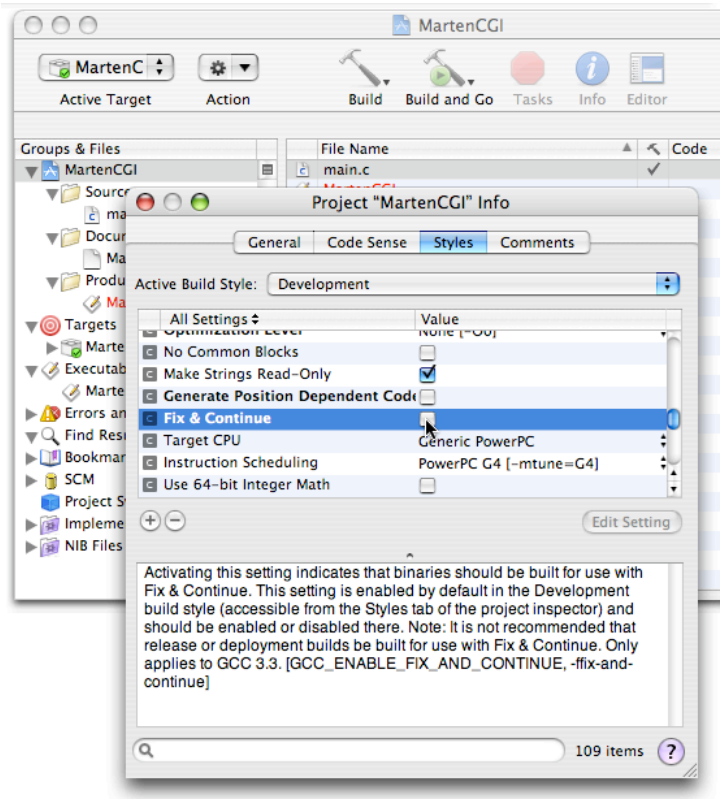
There are four settings that must be adjusted for this project. Select the MartenCGI project icon under Groups & Files and click the Info button on the toolbar. The MartenCGI project info window will open. Click the Styles tab and scroll down to the ZeroLink checkbox. Deselect ZeroLink.



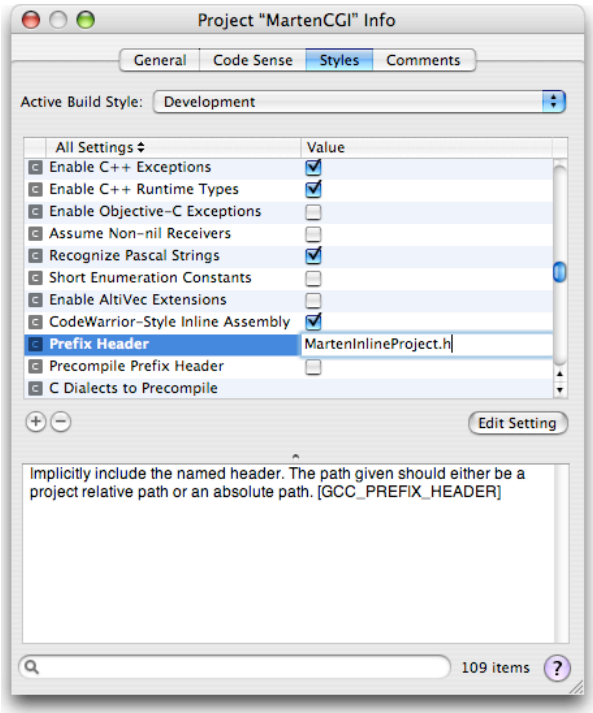
Next, name the product "testMarten.cgi". Be sure to spell it correctly, it must match the action attribute for the form in the "Show Form" universal.



We don't need Fix & Continue, so deselect that checkbox.



Finally, the Marten_Headers header file should be included in every file of the project, so type that in as the Prefix Header:

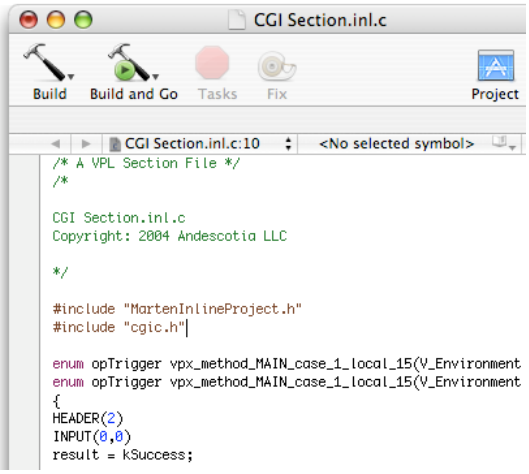


Modifying the exported C files

Next delete the main.c file from your project and add all of the C source code files to the Source group of your Xcode project. The files are:

CGI Project.inl.c
CGI Project.inl.h
CGI Section.inl.c
cgic.c
cgic.h
MacOSX CGIC.c
MacOSX CGIC.h
MacOSX_CGIC.c
MartenInlineProject.h

As it stands, the C files will not yet compile. First you must implement the workaround described in "Marten-Make Command Line Example" to fix the export bug for the project C file and header file (this bug has been fixed for the soon-to-be-released Marten 1.2). The necessary modifications are described on pages 11 and 12 of that document. After those changes have been put in, open the CGI Section.inl.c file. You must provide declarations of the functions contained in the file cgic.c, so add an include statement "#include "cgic.h"" somewhere towards the top of the file.



The Boutell CGIC library provides a MAIN routine which performs some setup and then calls a routine called "cgiMain". Therefore the exported MAIN in the file CGI Project.inl.c must be modified to be this cgiMain routine. Change the main routine definition prefix to "int cgiMain(void)" and add two declarations for argc and argv as shown:

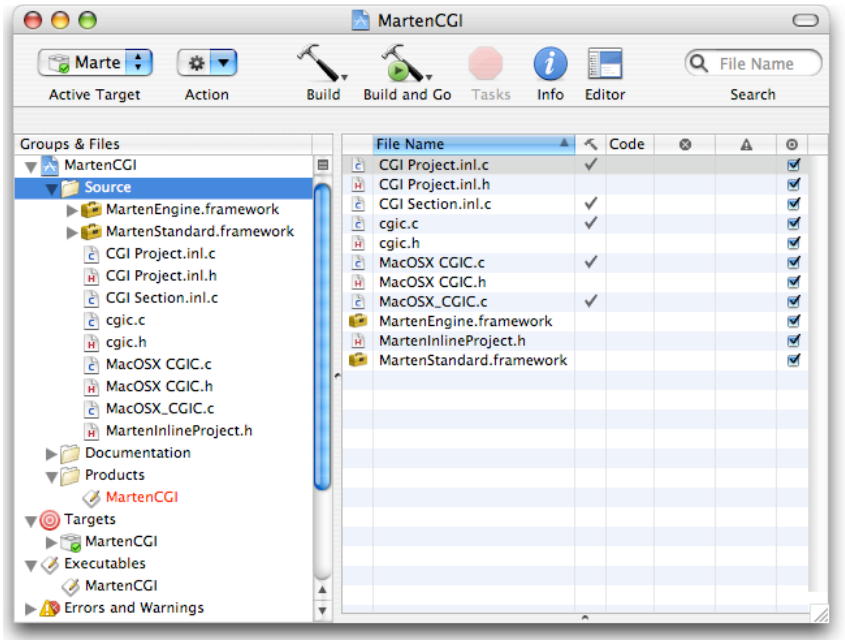
Before:

```
void init_project_CGI_20_Project(void);
void init_project_CGI_20_Project(void)
{
    if( gVPLCGI_20_Project_Environment == NULL ){
        gVPLCGI_20_Project_Environment = VPLEnvironmentCreate("CGI Project", kvpl_St

        load_project_CGI_20_Project(gVPLCGI_20_Project_Environment);

        VPLEnvironmentInit(gVPLCGI_20_Project_Environment, vpl_INIT);
    }
}

int main(int argc,char *argv[])
{
    init_project_CGI_20_Project();

    return VPLEnvironmentMain(gVPLCGI_20_Project_Environment,vpl_MAIN,argc,argv);
}
```

After:

```
void init_project_CGI_20_Project(void);
void init_project_CGI_20_Project(void)
{
    if( gVPLCGI_20_Project_Environment == NULL ){
        gVPLCGI_20_Project_Environment = VPLEnvironmentCreate("CGI Project", kvpl_S

        load_project_CGI_20_Project(gVPLCGI_20_Project_Environment);

        VPLEnvironmentInit(gVPLCGI_20_Project_Environment, vpl_INIT);
    }
}

int cgiMain(void)
{
    int     argc = 0;
    char    *argv = NULL;

    init_project_CGI_20_Project();

    return VPLEnvironmentMain(gVPLCGI_20_Project_Environment,vpl_MAIN,argc,argv);
}
```

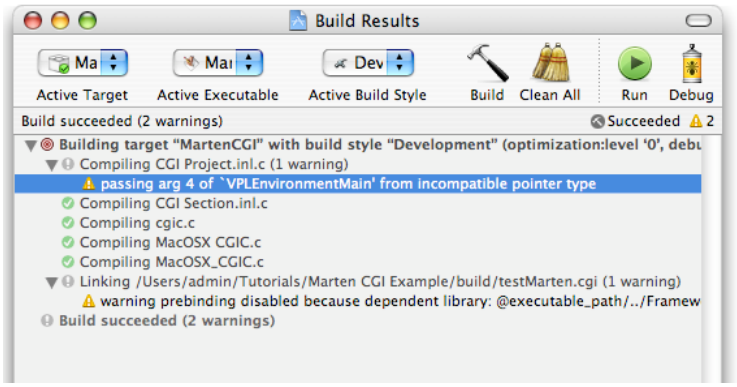
Finally add the MartenEngine and MartenStandard frameworks. The final set of source files for the project should look like:



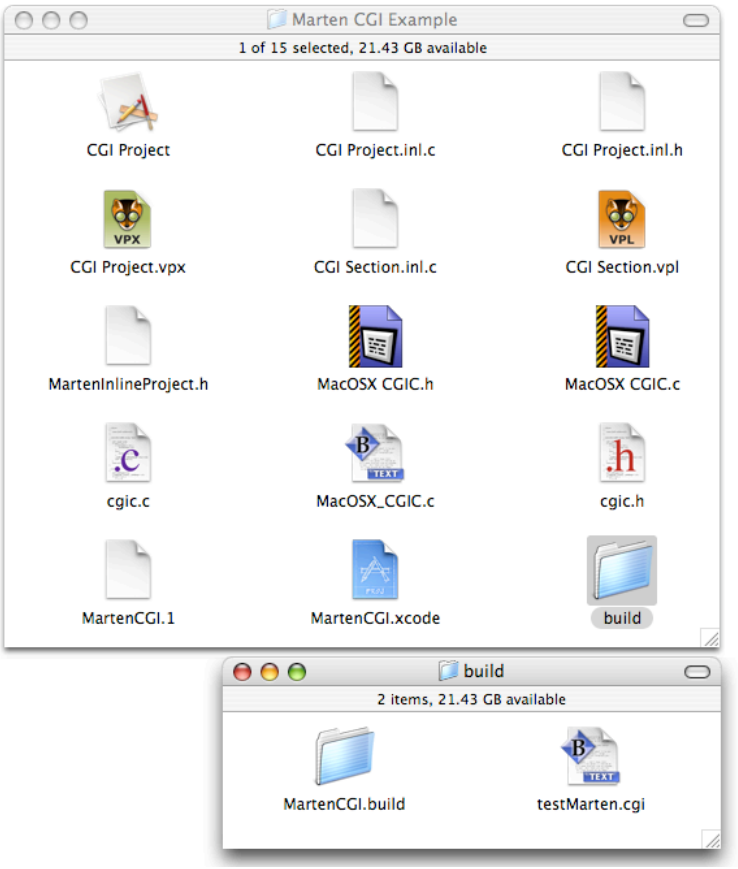
You are finished with creation of the project, now it is time to build the executable.

Building the CGI executable with Xcode

Select the "Detailed Build Results" command under the Build menu to open the Build Results window. Then click the "Clean All" button on tool bar to clear any previously created object code. Finally, click the Build button on the toolbar. The project will be compiled and linked with build results similar to:

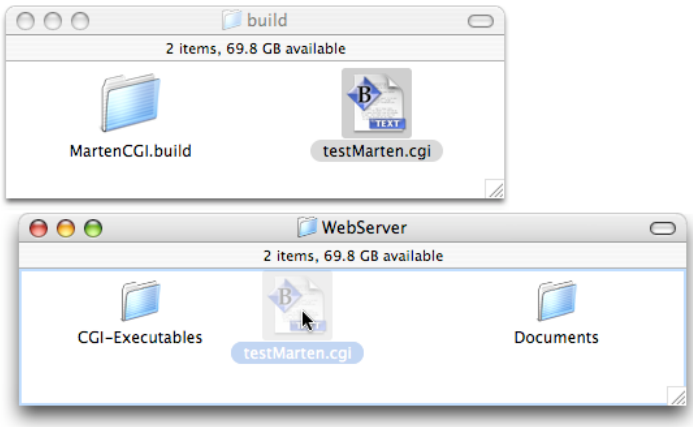


Quit Xcode. The set of files in the project folder should look match the following list and the actual CGI executable should be named testMarten.cgi and found in the "build" folder.



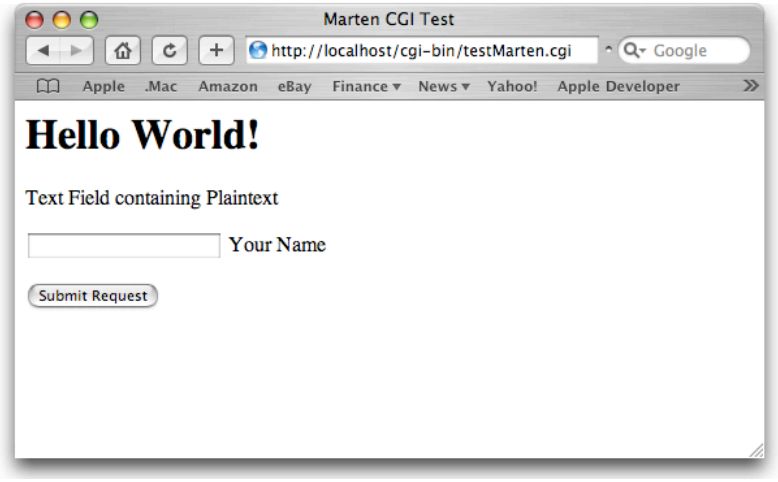
Installing the CGI executable

A CGI executable is launched by the MacOS X Apache server when it is invoked by the browser. For this to happen, the executables must be placed in a special location, the CGI-Executables folder found in the directory /Library/WebServer/. Copy the executable file "testMarten.cgi" into /Library/WebServer/CGI-Executables/. You can use the Option Drag technique as shown here:



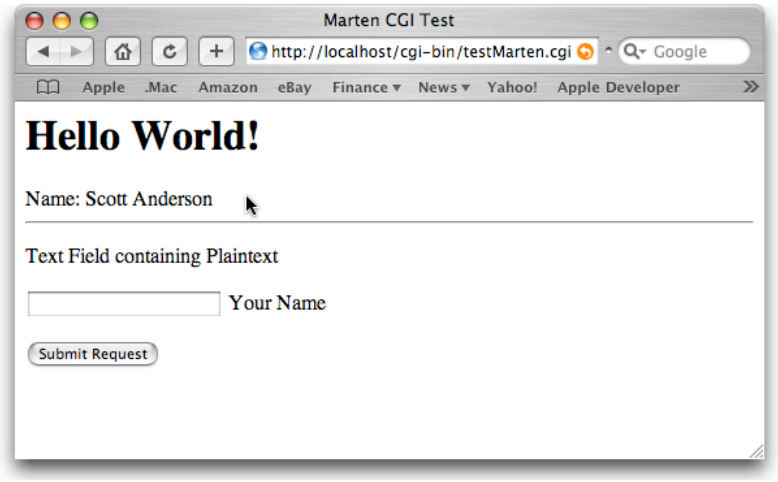
Running the CGI executable

To run your CGI executable, the Apache web server on your Mac must be running. To check that it is, open the System Preferences application and click on the Sharing icon of the Internet & Network group. Make sure the Personal Web Sharing service is checked and on. If it is, then launch a web browser (Safari is used for the following illustrations) and navigate to the URL "http://localhost/cgi-bin/testMarten.cgi". The CGI executable will be invoked and generate the web page:



Type your name into the "input text" box and click the Submit Request button.

The CGI executable will be invoked and this time, the "Handle Submit" universal will be called to display the contents of the text box.



This concludes the Marten CGI example. If you wish to create more CGI executables, we recommend designing a class library to make coding easier using the following guidelines:

1. Most of a web page consists of static information. Static content is best stored as a persistent. For example, the entire functionality of the Show Form universal is better implemented as a persistent containing an instance of a class whose attributes contain a "specification" (probably consisting of instances of other classes) of the web page. This class could contain a "Print Me" method that would take the specification and actually print it. Show Form would then just be a persistent operation followed by a data-driven class method.

2. The "specification" described above could also specify what method to call when some action is taken. In the Prograph language, such functionality is provided by injects and the "call" primitive. A good design should include some way to "call" methods by name and supply the appropriate parameters. An example of this kind of design is found in the Simple App example (cf: <Simple Method Call> etc.).