QSBuilder
Modular Quickstep Programming Guide

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions. See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

Model Number	Hardware Revision	Firmware Revision
All	All	All

WARNING: Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

TABLE OF CONTENTS

Blank Page

1.0 Overview

QSBuilder is a utility program that allows the user to extend the capability of the Quickstep Editor by allowing multiple Quickstep™ programs to be linked together to form a single compiled program that can be downloaded into a CTC controller. QSBuilder greatly simplifies the task of creating large machine control programs by making it easy to divide the overall control program into smaller programs or modules. Some of its other benefits are:

- Controller programs can be built as logical modules and then arranged in any order prior to compiling allowing for modular programming.
- Standard "library" modules can be built for common functions and then be used and combined as needed saving program development time.
- Allows very large programs to be downloaded into a controller. The QS Editor can only be used on program modules up to 64K in size (typically 1200 to 1500 statements). QSBuilder allows the user to combine up to 80 program modules into a single compiled program for downloading to a controller. From a practical sense, the only limitation to a program's length is the amount of program execution memory allocated in the controller (4096 steps in the 5100 and 2700 series controllers).
- QSBuilder uses a faster 32-bit compiler reducing compile times.
- Detailed compiler logging diagnostics
- Automatic Step Name resolution between modules

2.0 Installation

QSBuilder is distributed as a WINZIP file, qsbuilder.zip. Within this file are the needed files to automatically install QSBuilder, manually update QSEditor, and sub-folders containing the sample code found in this manual.

To install:

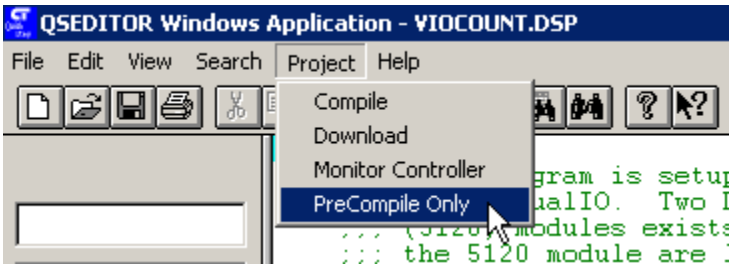
1. Create a temporary directory for qsbuilder.zip extraction.
2. Using WINZIP extract qsbuilder.zip to the temporary directory. Folders will appear in a sub-directory called QSBuilderTmp.
3. Run the setup.exe program in the QSBuilderTmp sub-directory and follow its instructions for installation. It is a standard install program. A DLL file called qscompiler.dll will be installed in your windows system32 sub-directory as part of the installation process. Make sure you have Window's installation rights prior to attempting installation.
4. QSBuilder is now installed and accessible from the Start->Program->CTC menu's.
5. The Quickstep Editor must now be updated. First backup your existing qsedit21.exe file found within the Quickstep Editor sub-directory. Next, from the

installation directory of QSBuilder, typically “Program Files\QSBuilder”, you will find a sub-folder called ‘quickstep’. Within that folder is a new ‘qsedit21.exe’ that should be used to replaced the one you just backed up. This adds the ‘PreCompile Only’ menu option to the Quickstep Editor.

QSBuilder is now installed and is available from the ‘Programs’ menu.

3.0 Operational Details

QSBuilder requires a unique file, called a Precompiled file (.PRE). The Quickstep Editor generates this file from the ‘PreCompile Only’ menu option, instead of using the standard ‘Compile’ menu item.



A new Editor is required to utilized this feature and is contained in builds after and including, V2.3, April 2003. The present version of the Editor can be found by selecting the ‘Help’ menu, followed by the ‘About’ menu option.

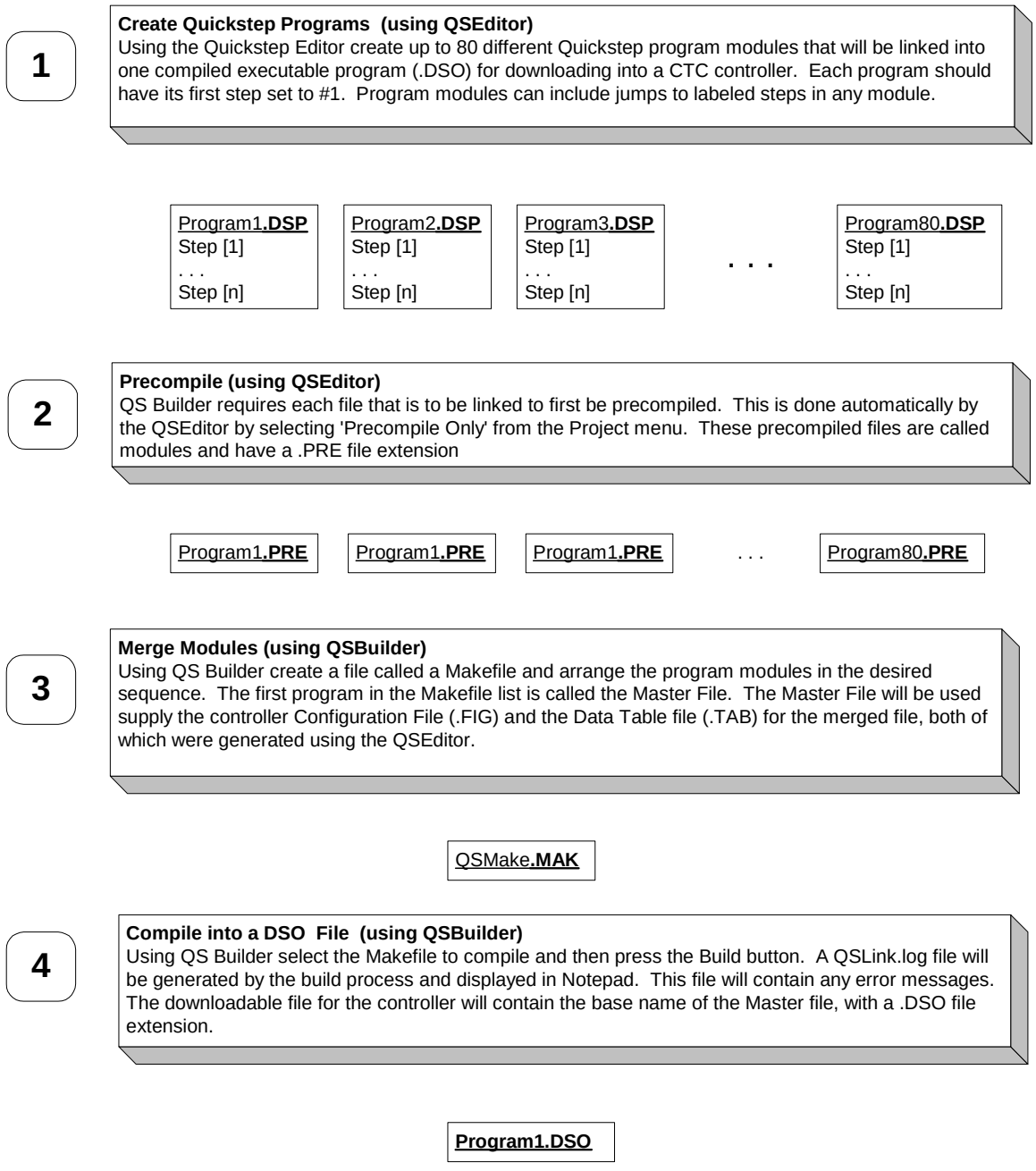
A Precompiled file is generated for each Quickstep Program Module desired. It is suggested that steps in each module begin at [1] and no direct branching to a numeric step be done, only symbolic labels for step names should be used. Alternatively, if breaking up an existing program, each module can start at the last step of the previous module, plus one. Thus if module 1 ended at step 450, then module 2 must start at 451 (or [1]) otherwise an error will be generated. When a module starts at step number [1] then it will automatically add the last step number of the previous module to the step of the current module. Thus if breaking up an existing program and the module step began at [451], simply change it to a [1], leaving the other steps as they are [452], [453],... It will create a gap but will link correctly. As modules are loaded Step Names assigned to a particularly step have their step numbers automatically re-numbered, as required. This is why you can start each module at step [1] as long as you use step label names for your branch operations and not direct step numbers. Up to 80 modules may be used when linking a program.

Branching from one module to another can easily be done, simply by naming the step in the other module. The PreCompiler will not declare an error since validation of step names is not done until modules are linked. Duplicate step names are not allowed.

The first module referenced during a build process is known as the Master module. This module is responsible for supplying the Configuration file (controller selection, .FIG) and

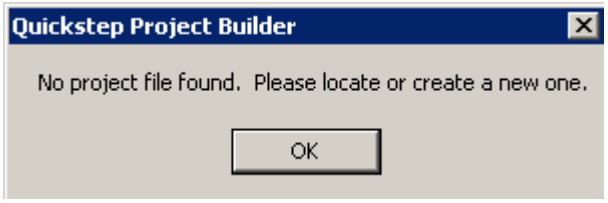
the any Data table (.TAB). Data tables and Configuration files that exist in other modules will be ignored.

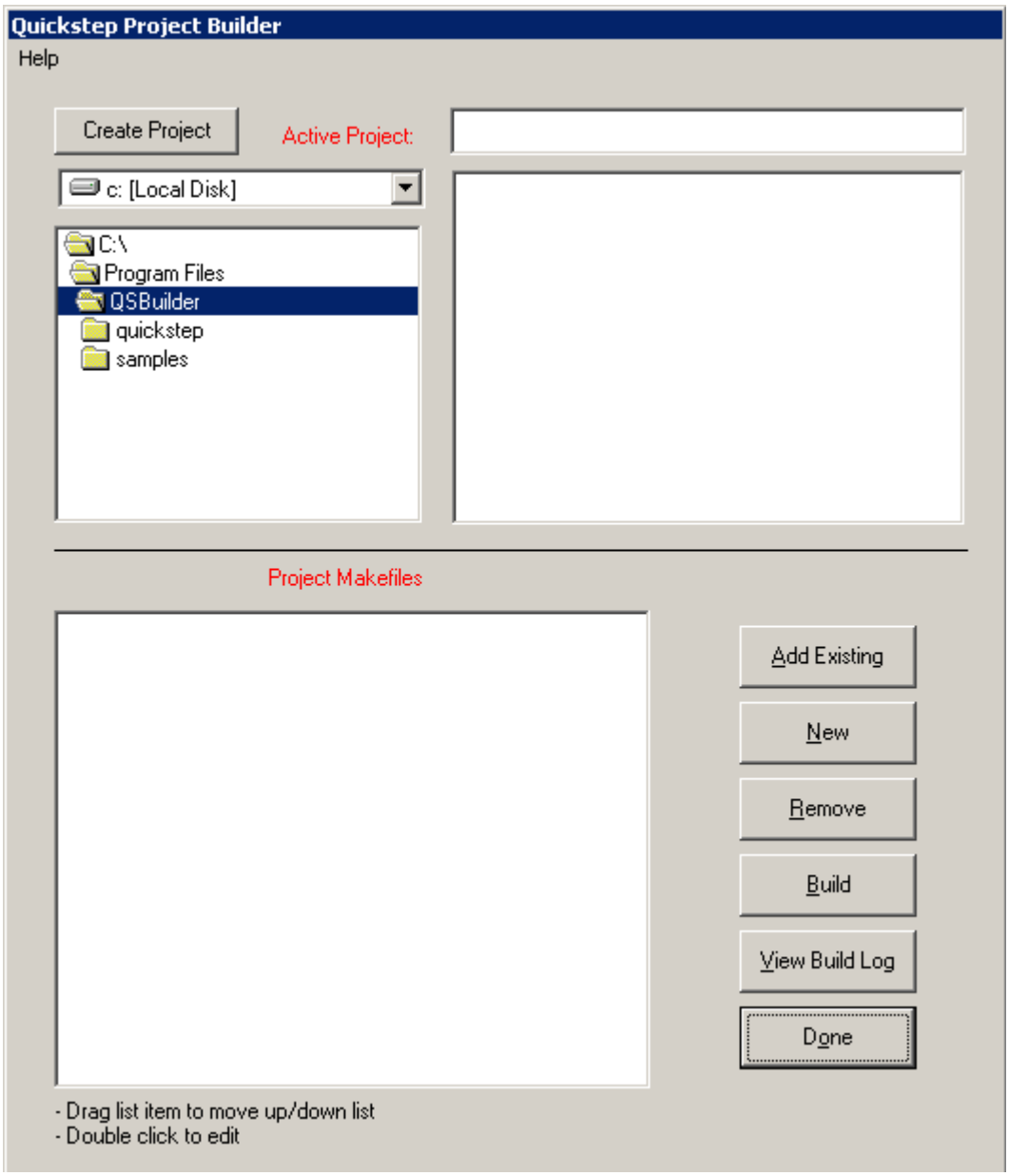
3.1 Process Flow Summary



4.0 Project and Makefile Creation

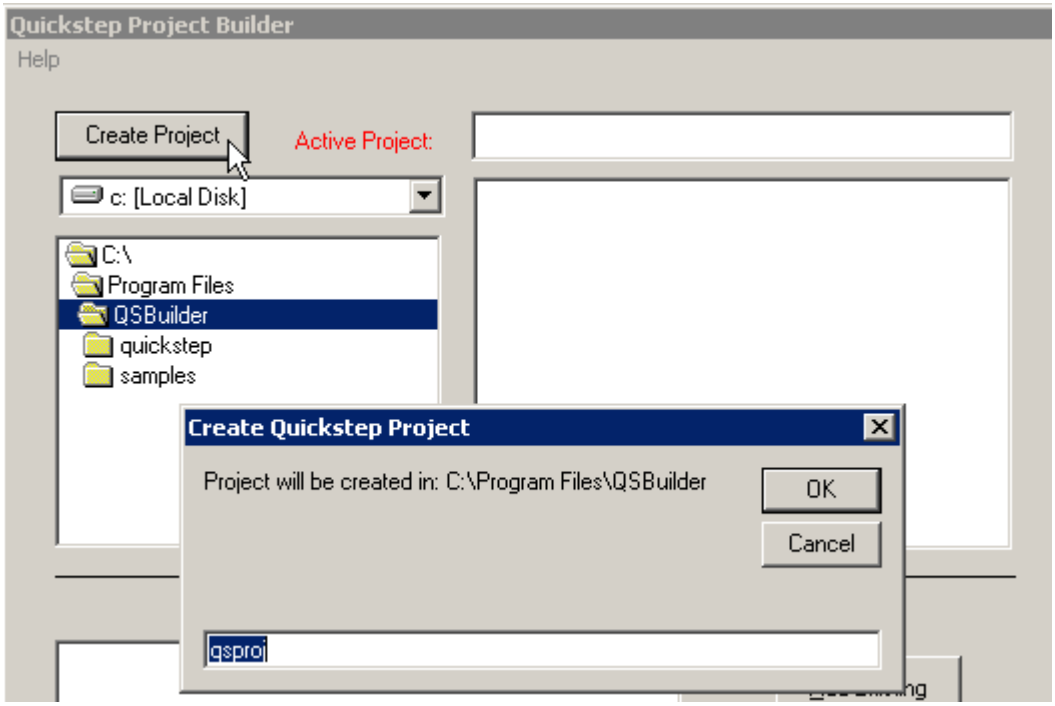
This section will review the creation of a Project and Makefile for compiling/linking Quickstep modules. The files residing in the samples\VirtualIO subdirectory are used in the following example. When QSBuilder is executed for the first time no projects exist.



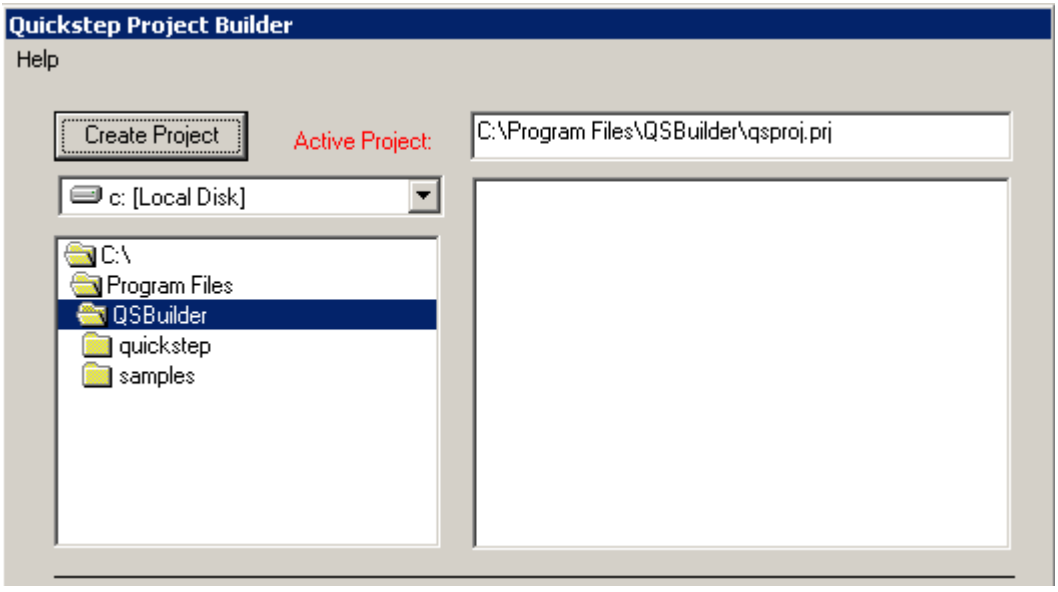


Using the built-in File Manage utility, select the desired directory for creating the project. Then select the 'Create Project' button. A project can be though of as a group of build environments for different Quickstep Programs. In many cases you may only have a single project, although if the program is shared among users, each may have a project file with their own name.

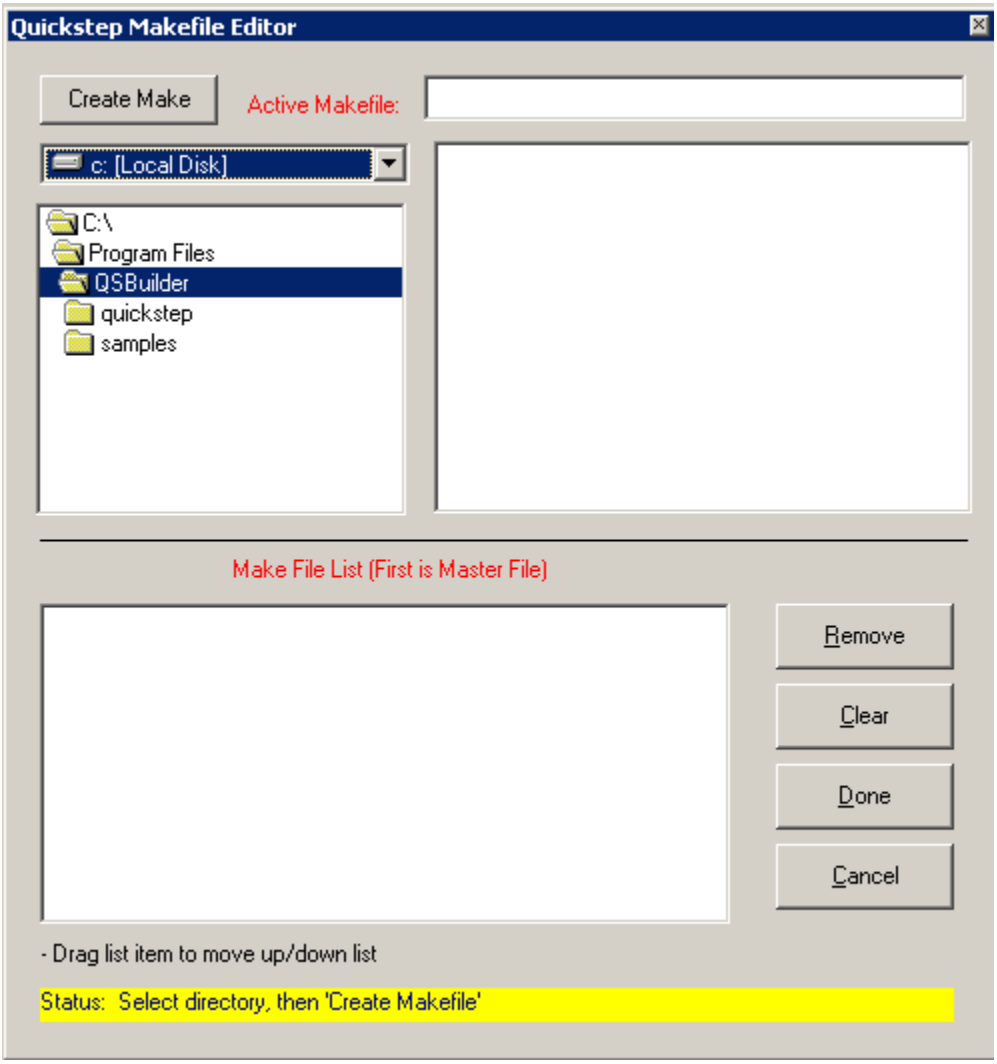
A pop-up dialog box will appear requesting the base name of the project:



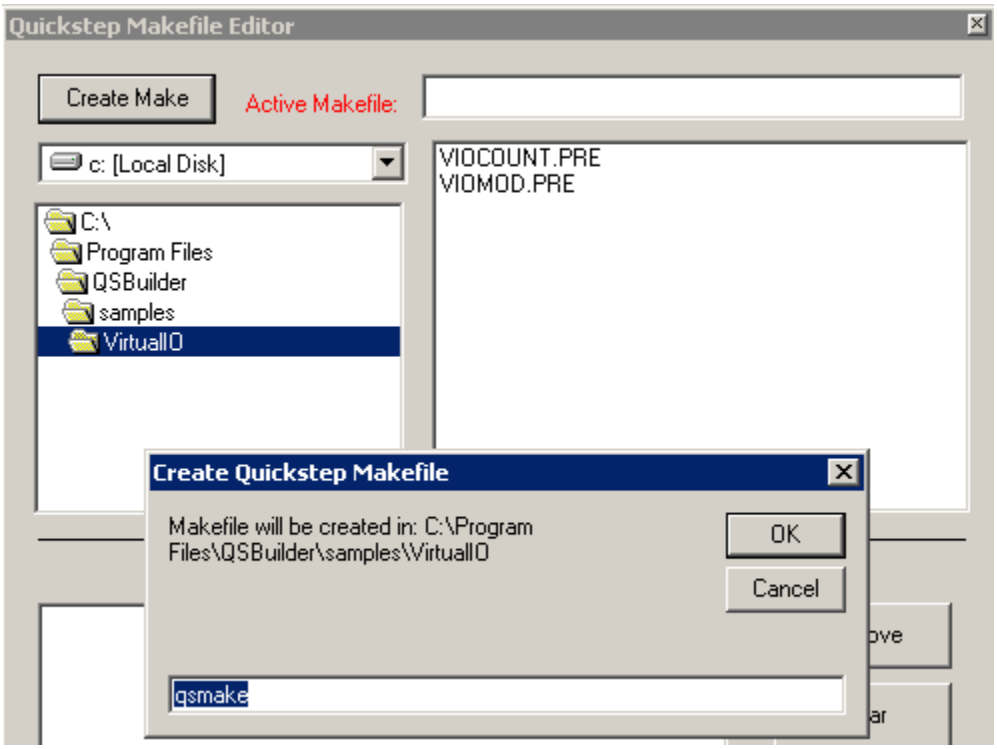
In this case the default name of 'qsproj' will be created in the C:\Program Files\QSBuilder directory.



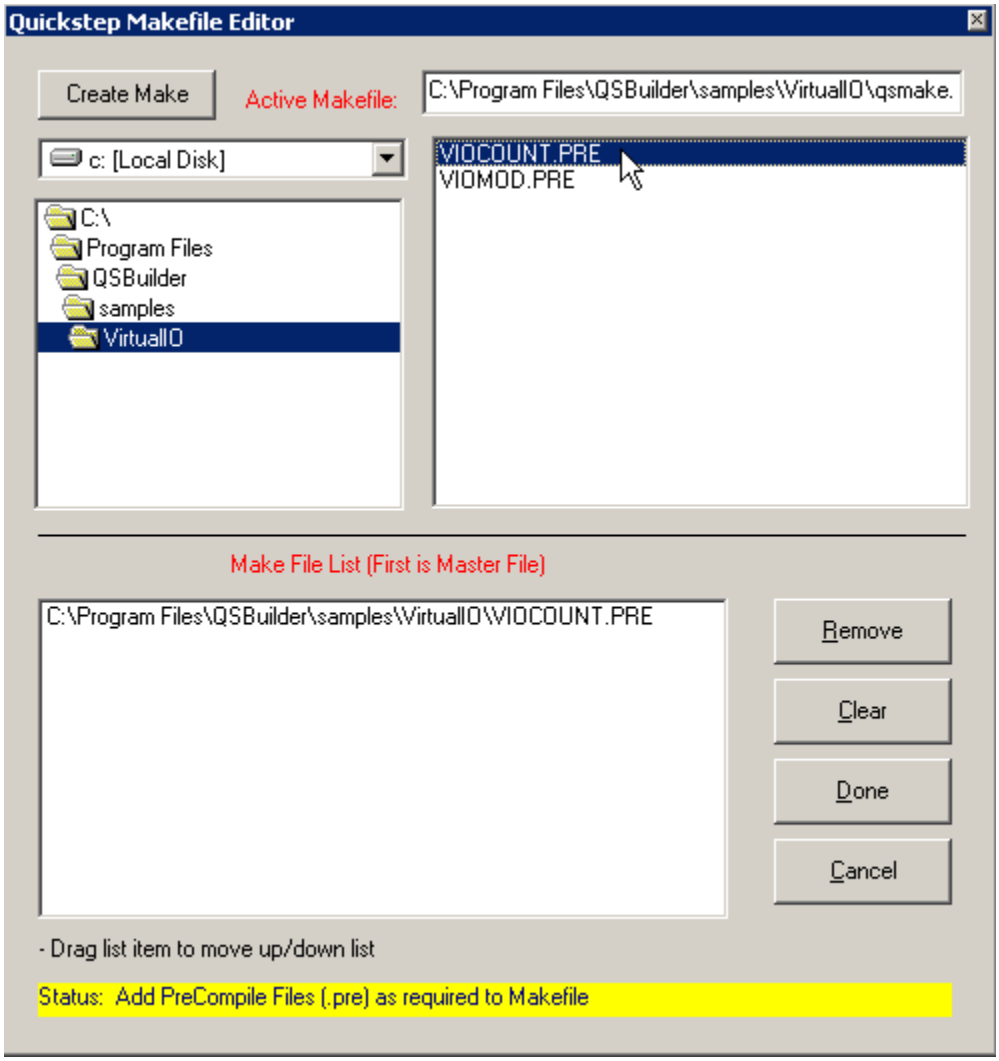
Next a Makefile must be created. Each Makefile represents a collection of modules that are required to create a single .DSO Quickstep executable file. This is the file downloaded to the controller. As with the Project file, some programmers will only have a single Makefile. To create the first Makefile select the 'New' button. To edit an existing Makefile simply double-click the entry in the list.



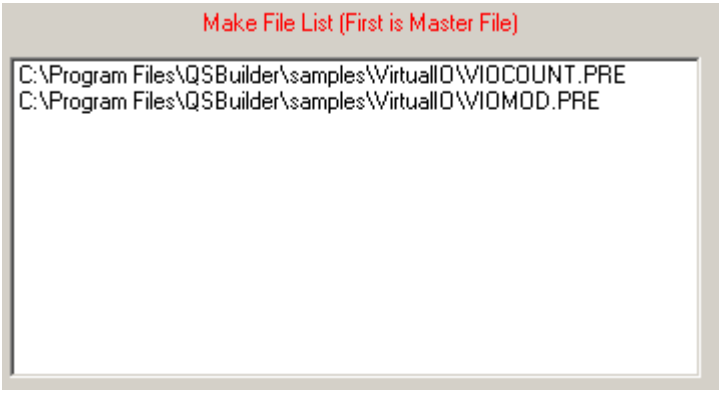
The Makefile Editor will appear. Note the yellow 'Status' line at the bottom of the screen. It will tell you the mode of the editor. Currently it is requesting a directory to be selected for the creation of a Makefile. The Makefile will contain a reference to each of the .PRE, Pre-Compile, files generated from the Quickstep Editor. Typically the Makefile is stored in the folder with the .PRE files, but it is up to the user.



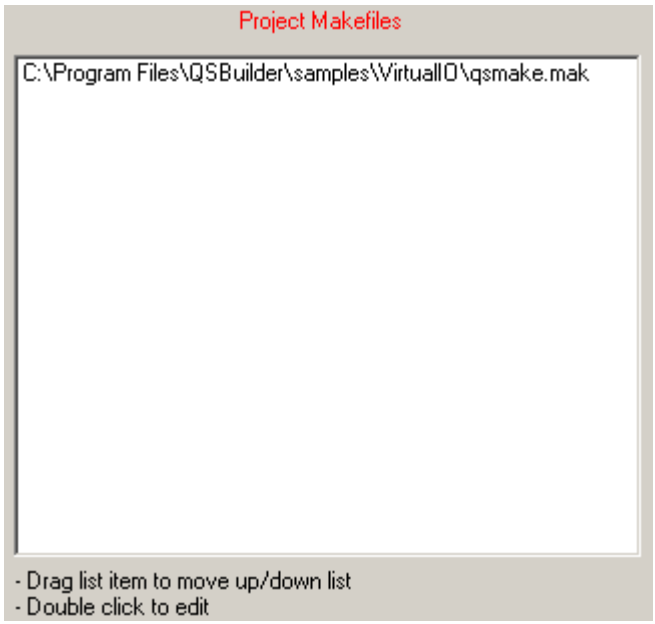
In the above example the Makefile is being created within the sample directory shipped with QSBuilder. Note the .PRE files that are also available. They will later be added to the Makefile as files to be linked into a single .DSO output file. The PreCompile files are added by clicking on them in the order they are to be linked. The first one is known as the Master file. The Master file is the file responsible for the controller Configuration (.FIG) and Data table (.TAB). The Master file is always the first file in the list. To change the order of files simply drag the file up/down the list as required. Highlighted entries may also be removed from the list using the 'Remove' button. Removing all entries deletes the Makefile. Below is an example of adding the Precompile file, VIOCOUNT.PRE.



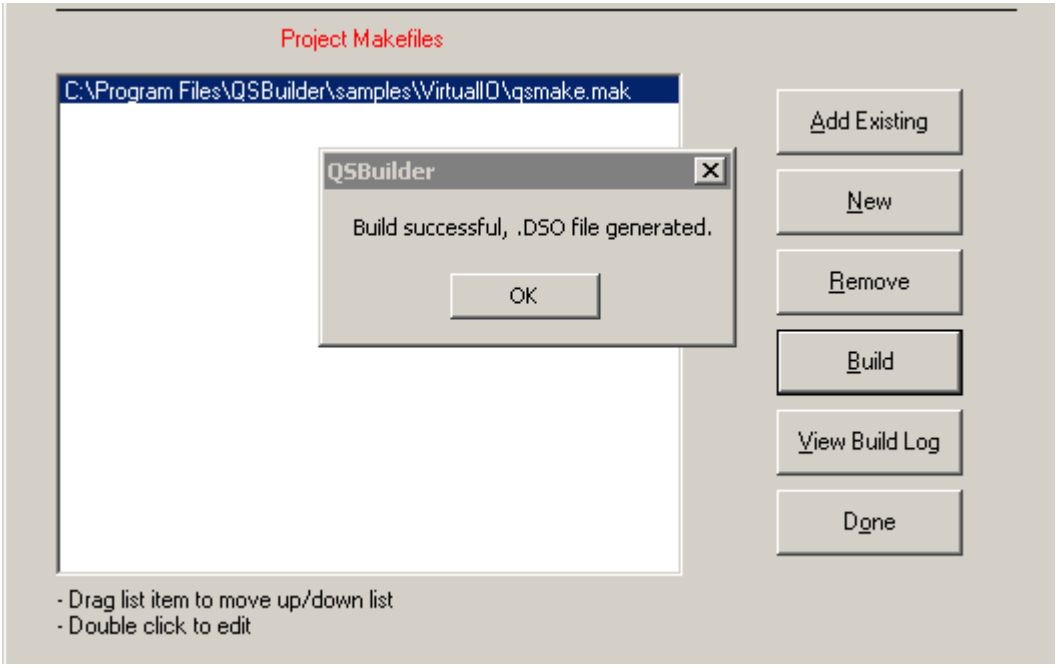
VIOMOD.PRE can then also be added by clicking it:



When done adding files, select the 'Done' button to permanently store the information to the hard disk. An entry for the new Makefile will also be placed in the Project Makefile list and you will be returned to the Project Builder Screen.



From the Project Builder Screen, select the 'qsmake.mak' file to highlight it, then the 'Build' button in order to Compile and Link the desired files. The output log generated is shown in the next section.



5.0 Module Build Example

Below are two program modules, included in the samples\VirtualIO sub-directory after installation. Each program begins at step [1], with VIOCOUNT.DSP being the master file. Symbol information is unique to both this file and VIOMOD.DSP. To avoid having conflicts with symbol files, it is suggested that the Master file contain the symbols for the entire program. Unused steps in other modules can be declared as 'Undefined', type 65536. Typically common symbols, with no step names, would be created, then the file copied to the base name of the other modules. For example VIOCOUNT.SYM could be copied to VIOMOD.SYM, prior to adding step names. Just remember, in the pre-compile stages, prior to linking, each I/O resource symbol defined is unique to a module's own symbol file. Unique resources can be added to each module without affecting any other modules. Copying the Master symbol file is just a suggestion to avoid a lot of entering of duplicate symbols, for each module.

5.1 VIOCOUNT.DSP

```
[1] init
   ;;; This program is setup to interface with a single remote 5100
   ;;; via VirtualIO.  Two Digital Input (5110) and two Digital Output
   ;;; (5120) modules exists in the remote controller.  The Outputs of
   ;;; the 5120 module are looped back to the inputs of the 5110.
   ;;; .
   ;;; Upon power up we must see what state we were in.  This
   ;;; may simply be a program restart and the connection could
   ;;; already be active.
   ;;; .
   ;;; Check the Virtual IO status
   ;;; 1  = ready
   ;;; -1 = requires full initialization
   ;;; -2 = already initialized, just need to initiate connection
   ;;; .
   ;;; Also note that a Fault Handler is installed in the first
   ;;; step to monitor for communications failure.  The FaultMaskRegister
   ;;; must be set after the FaultStepRegister, otherwise the handler
   ;;; will be invoke for all faults (default).
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 0 to sum
   store 0 to cyclecount
   store 8 to FaultStepRegister
   store 32 to FaultMaskRegister
   if v_status=1 goto execute
   if v_status=-1 goto v_setup
   if v_status=-2 goto go_online

[2] v_setup
   ;;; Initialize Virtual IO for operation, remote IP address
   ;;; is 12.40.53.197.  Also set high priority mask for
```

```
   ;;; Digital Inputs 1-8.  Init the register remap capability
   ;;; to the 23000 block.  192 registers will be present.
   ;;; To disable this feature simply set the v_remapstart register
   ;;; to a 0.  Finally create the server register block by writing
   ;;; a 1 to v_command.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store 12 to v_IPA
   store 40 to v_IPB
   store 53 to v_IPC
   store 197 to v_IPD
   store 23000 to v_remapstart
   store 1 to v_command
   store 50 to v_IOscanrate
   goto Next

[3] go_online
   ;;; Now initiate the actual connection to other 5100 now and add its
   ;;; sticks to our buss.  Also note the high priority mask
   ;;; was set to the Digital Inputs.  Do not use the high
   ;;; priority mask on fast acting, repetitive signals.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store 255 to v_DI_MASK
   store 2 to v_command
   goto Next

[4] wait_for_virtualIO
   ;;; Loop here until we finish connecting to other 5100
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   if v_status=1 goto execute
   delay 100 ms goto wait_for_virtualIO
```

Note that in step [1] and [4] there is a branch to a step called 'execute'. This step does not exist in this module therefore if we were to compile just this module an error would result. VIOMOD.DSP contains 'execute'.

5.2 VIOMOD.DSP

```
[1] execute
   ;;; Now begin main program execution.
   ;;; .
   ;;; This program sets a digital output, then looks for that
   ;;; output on its input, timing how many milliseconds it
   ;;; took (time).  Begin by reading the current output value
   ;;; and incrementing it by one.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store Reg_10101 + 1 to outputVal
```

```
   if outputVal <=65535 goto monitor_change
   store 0 to outputVal
   goto Next

[2] monitor_change
   ;;; Now wait for the set digital output to be read as an
   ;;; active input.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store outputVal to Reg_10101
   store 0 to time
   if outputVal=reg11101 goto IO_changed

[3] IO_changed
   ;;; Save the number of milliseconds took for Quickstep to
   ;;; set and detect the input change to "lastTime".  Also
   ;;; set the largest time it took so far in "maxTime" and
   ;;; maintain a running average in "avgTime"
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store time to lastTime
   store cyclecount + 1 to cyclecount
   store sum + time to sum
   store sum / cyclecount to avgTime
   if time <=maxTime goto execute
   store time to maxTime
   goto execute

[4] FaultHandler
   ;;; This step is invoked should a fault occur, such as a
   ;;; network disconnect.  The FaultMaskRegister controls
   ;;; under what circumstances the handler is invoked.  This
   ;;; example is very simple.  It basically shuts all the
   ;;; local outputs off and sets a flag in FaultFlag that
   ;;; has no purpose.  Note that no other tasks will be running
   ;;; in the system nor can this task fault when the handler
   ;;; is invoked.
   ------------------------------------------------------------------------
   <TURN OFF ALL DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 1 to FaultFlag
   delay 3 sec goto ClearFault

[5] ClearFault
   ;;; Now attempt to recover from the fault by issuing a RESTART
   ;;; command
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 2 to FaultFlag
   store 5 to FaultClearRegister
   goto FaultHandler
```

Note that a 'goto Next' in the last step of a previous module will cause a branch to the first step of the following linked module.

Results from a build:

5.3 Qslink.log

```
QSBuilder V1.0, April 14, 2003

Reading DSP/MAK program from file "C:\Program Files\QSBuilder\qsmake.mak"...

Reading configuration parameters from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.fig"...

Warning:  Unrecognized parameter:  "stepping motors"

Warning:  Unrecognized parameter:  "0"

Read configuration parameters from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.fig".
Reading data table from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.tab"...
Read data table from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.tab".
Reading DSP/PRE program steps from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.PRE"...
Reading DSP/PRE program steps from file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOMOD.PRE"...
Read file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.PRE" [9 steps, 1 output changes, 39 instructions].
Compiling source file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.PRE"...
Checking labels...
Saving cross-reference map to file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.xrf"...
Saved resource cross-reference map in file "C:\Program Files\QSBuilder\samples\VirtualIO\VIOCOUNT.xrf".

********** COMPILE/LINK COMPLETE **********

Successful compile: 255 bytes program, 40 bytes data table (0% RAM used).
```

6.0 Terms Summary

Term	*Description*
Build	The process by which QSBuilder, links all the modules found within the Makefile into a single, .DSO, downloadable controller program.
Configuration File	The standard Quickstep Editor .FIG file that contains information specific to the controller type an application program is targeted to execute on.
Data Table	The standard Quickstep Editor .TAB file that contains program data table values.
Module	.PRE file generated by the Quickstep Editor PreCompiler.
Makefile	Text file, with a .mak extension, that lists all the modules used to create a downloadable image for the controller
PreCompile, PreCompiler	The interim process completed by the Quickstep Editor, prior to a complete Compile. The PreCompile resolves all I/O resources, but not branch label names. A file with an extension of .PRE is generated. Each .PRE file is known as a Quickstep Module. Modules may be grouped together by QSBuilder Makefiles to generate a downloadable image for the controller.
Project File	Text file, with a .prj extension, that contains a list of Makefiles, each representing a different build environment for a controller application program.
QSLINK.LOG	QSBuilder compiler output file which contains a copy of the last build process. Error messages will be contained here.
.DSO	File extension of downloadable controller programs, in binary format.
.DSP	File extension of a standard Quickstep Editor program, prior to any compiling. It is the source code. Multiple .DSP files can each generate a .PRE file, that is finally linked by the QSBuilder into a downloadable controller program.
.FIG	Standard Quickstep Editor controller Configuration file
.MAK	QSBuilder Makefile extension, refer to Makefile.
.PRE	File extension generated when a .DSP file is pre-compiled by the QSEditor.