CONTROL TECHNOLOGY CORPORATION

5300 ‘C’ User Programming Guide

5300 ‘C’ User Programming Guide

Blank

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.

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
5300	All Revisions	>= 5.00.89

TABLE OF CONTENTS

CHAPTER

1

Introduction

An advanced programming capability is supported by the Blue Fusion series of controllers, over and above the standard Quickstep™ environment. This allows independently compiled 'C' programs to be loaded into memory for execution alongside Quickstep programs. CTC recommends that only the most advanced programmers should consider the use of the features described in this document. This restriction is necessary due to the fact that extensive control is given to the 'C' (simple C++ is also supported) user functions and improper use can result in an unsafe controller. This document describes the environment available within the 5300 controller. In most cases previous 5100/5200 ‘C’ code should execute on the 5300, once re-compiled.

The 5300 ‘C’ API is a superset of the API used on the 5100/5200 and is the architecture upon which QuickBuilder™ executes, thus it is well tested. In theory the ‘C’ API can be used in conjunction with QuickBuilder but this is not recommended. QuickBuilder provides an integrated capability where ‘C’ functions can be included with your high level code providing a much cleaner environment. It is also possible to use the API described within this document to program the entire controller in ‘C’. Given QuickBuilder actually generates ‘C’ code that complies to this API, we again recommend using QuickBuilder to expedite project development.

6 MB of dynamic memory resides in the 5300 controller where user programs can be loaded and executed. Currently programs reside on the flash disk in the */_system/Programs* directory. Multiple programs may be resident, each loaded/unloaded by Quickstep and script control running the ‘`run userprogram <filename>`’ script command.

'C' and ‘C++” programs have access to virtually all the resources of the 5300 and more can be added as is needed. Currently access to serial communications, UDP packets, TCP virtual serial ports, motion control, analog, digital, register, variants, step logic, etc. is available. User functions can be called upon initialization, periodic tics, as an independent Quickstep step, and even as a totally separate thread. An independent memory and heap area is used, including the 'C' library that is linked. Virtual function calls are used to expose the Quickstep environment.

Additional functionality will be added to the 'C' programming environment, upon review of customer requests. CTC believes the growth of this aspect of the product is best initiated by the user community. If you believe that you need features not described within this document, contact your CTC regional sales representative.

Compiler

The compiler is available from MicroCross's website and is based on GNU, at http://www.microcross.com/html/visual_x-tools.html. Only the gcc compiler V3.4 is supported by to the 5300, for the Atmel ARM9. . Various manuals documenting the compiler are available from MicroCross and RedHat at http://www.redhat.com/docs/manuals/gnupro. Since the compiler is an open source product MicroCross is providing support for the tools for their fee. If you do not wish to have tools support, the compiler is available free of charge from Control Technology's web site, http://www.ctc-control.com/customer/idxdownloads.asp as a zip file. The zip file is quit large, around 700 Meg, but does contain all necessary files, including the development source code.

No compiler support is provided by CTC, only MicroCross on a fee basis.

Full floating point capability is supported by the 5300 although the printf, sprintf, and vsprinf functions that reside within the GNU libraries are not used, and have been slightly modified. A full build environment, with makefile, is also available for 'C' User function development. They are supplied as a .zip file, *5300UserC.zip*, for installation in the *5300UserC* directory.

Distributed 'C' Files:

The CTC 'C' Programming zip file contains a number of files. The 'C' source files are located in the *5300UserC/Source* subdirectory, header files in *5300UserC/Headers*. Files of interest are:

CoreFunc.c – CTC provided file, not to be modified, which provides a clean interface to the controller OS exposed virtual function table. A table of pointers to core OS functions is maintained and isolated by this module.
UserStart.c – CTC provided file, not to be modified, which contains printf/sprintf substitute, memory allocation routines and some other general functions.
UserApp.c – User program and sample code, contains 'C' main function entry point and is the file that the user will modify.
uTable.h – Main include file and definition file for virtual function calls and user/Quickstep interface structures.
aload.sr1 - Output file of sample build. When placed in the controller */_system/Programs* subdirectory, it may be loaded and executed using the `run userprogram` script command.

makefile – Rules file for building user ‘C’ application programs.
build5300.bat – Example build file on how to modify the file environment to build an sr1 file for the 5300.

Features

'C' user programs are available to provide advanced functionality. They may be used in a number of scenarios to enhance the Quickstep language and/or work independently. Some examples are:

Resource Filters

A resource filter is a 'C' function that will be called whenever a read or write is made to a resource. The actual value is presented to the 'C' function and it may simply return the same value or modify it as desired. Resource filters can be installed for any Data Register, Data Table access, Analog Input/Output values, and/or Motion values. Filters can provide universal conversions such as scaling, complicated floating point math calculations, maintaining your own 'C' tables and arrays, or virtually any other manipulation that is required while intercepting a Quickstep or Network read or write operation to a resource.

Tasks

'C' functions can be called just like Quickstep steps, accessing the same resources, manipulating data, custom communications protocols, string manipulation, etc. Access to the standard Quickstep task round robin loop is available. User functions can be called once per task loop, or as super tasks, after each Quickstep step.

Communications

User functions have complete access to both the standard serial port raw data and TCP virtual serial ports (see Lantronix COBOX terminal server), and UDP raw socket packet interface. This allows for easy manipulation of string data for placement in registers and the addition of custom communication protocols.

Motion Control (TBD)

User functions have complete access to the same function calls made by Quickstep to add complicated motion control algorithms. Filters can be added to Motion registers to ease the calculation of arc, build tables on the fly, etc.

Program Control

Just as multiple Quickstep programs can reside and be dynamically loaded from the flash disk, User Programs can also be changed dynamically.

Expansion

As new features are needed, the 'C' User Function interface will be expanded with additional capabilities added, based on real world input. Library functions are expected

to be made available for standard conversions, filter examples, motion control, and advanced communications.

CHAPTER

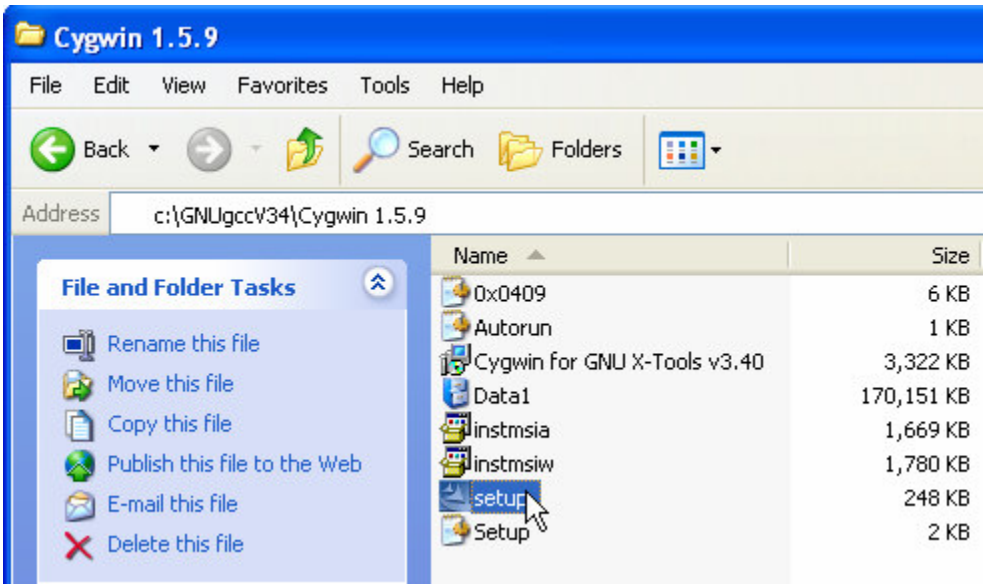
2

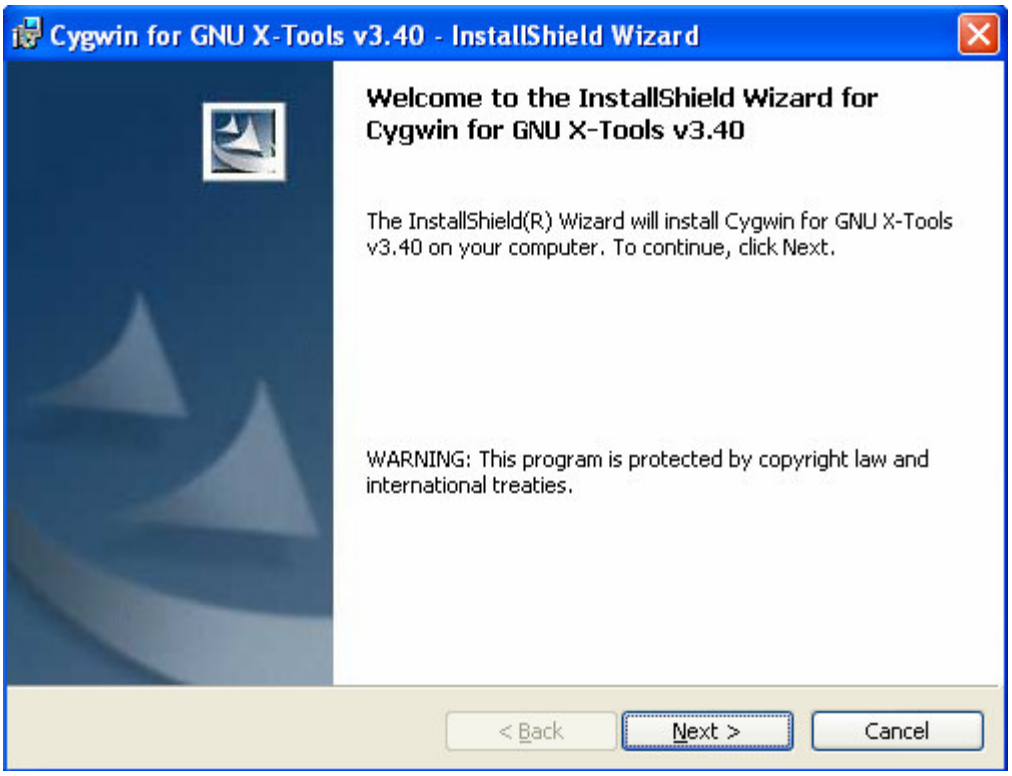
Tools

The compiler tools are based on open source, industry standard GNU gcc, freely distributable and extremely stable. This provides a very cost effective development environment while benefiting from a large community of developers. The instructions below detail use of the downloadable zip file, available on CTC's web site. It is assumed that if the tools are purchased from Microcross the user will follow their instructions.

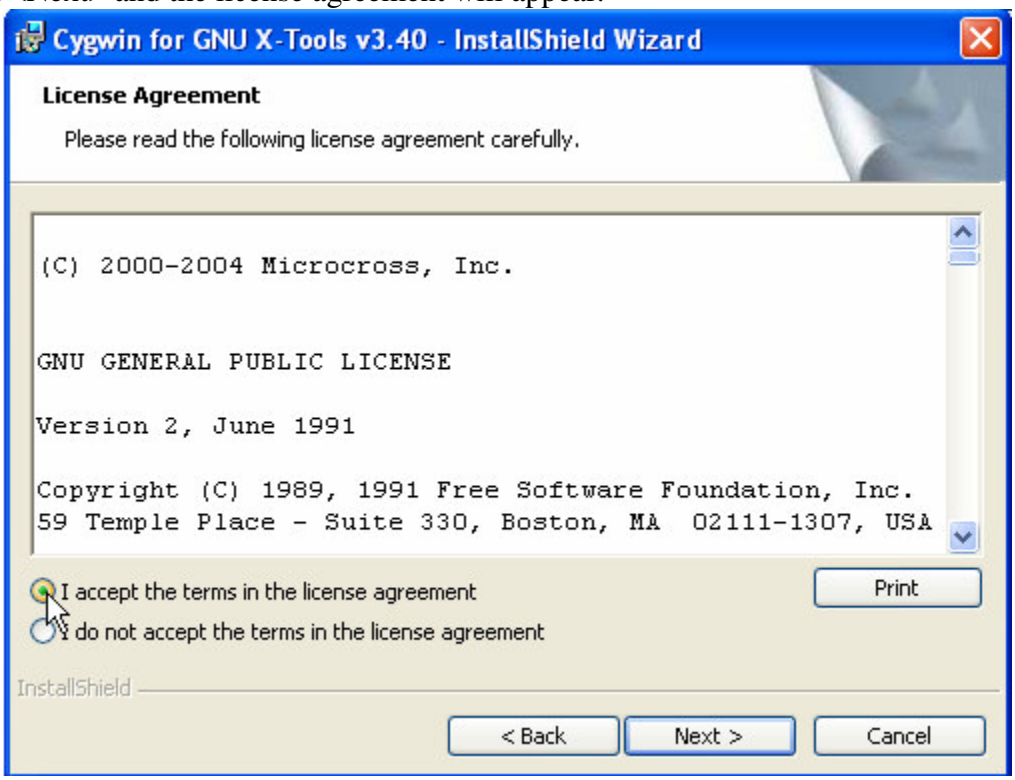
Cygwin v1.5.9 Installation (compiler)

Insert Microcross CD labeled "CYGWIN" (v1.5.9) or extract the downloaded zip file to a directory of that name. The welcome message will appear if autorun starts properly, otherwise double-click the setup application and the following Welcome screen will appear.

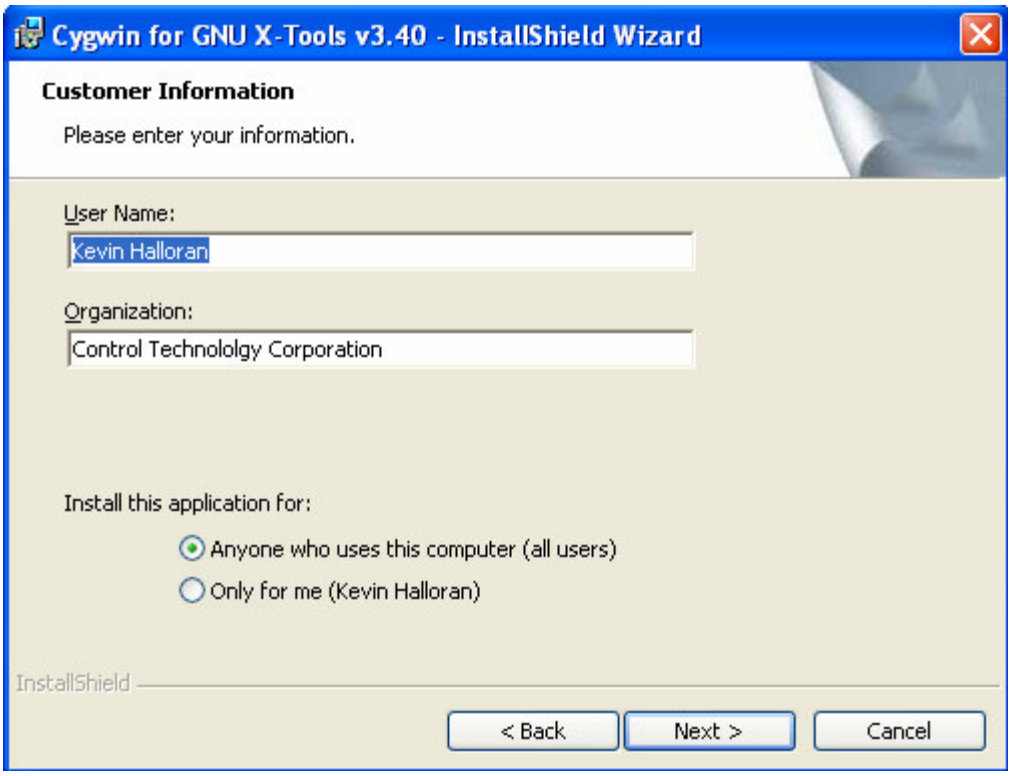




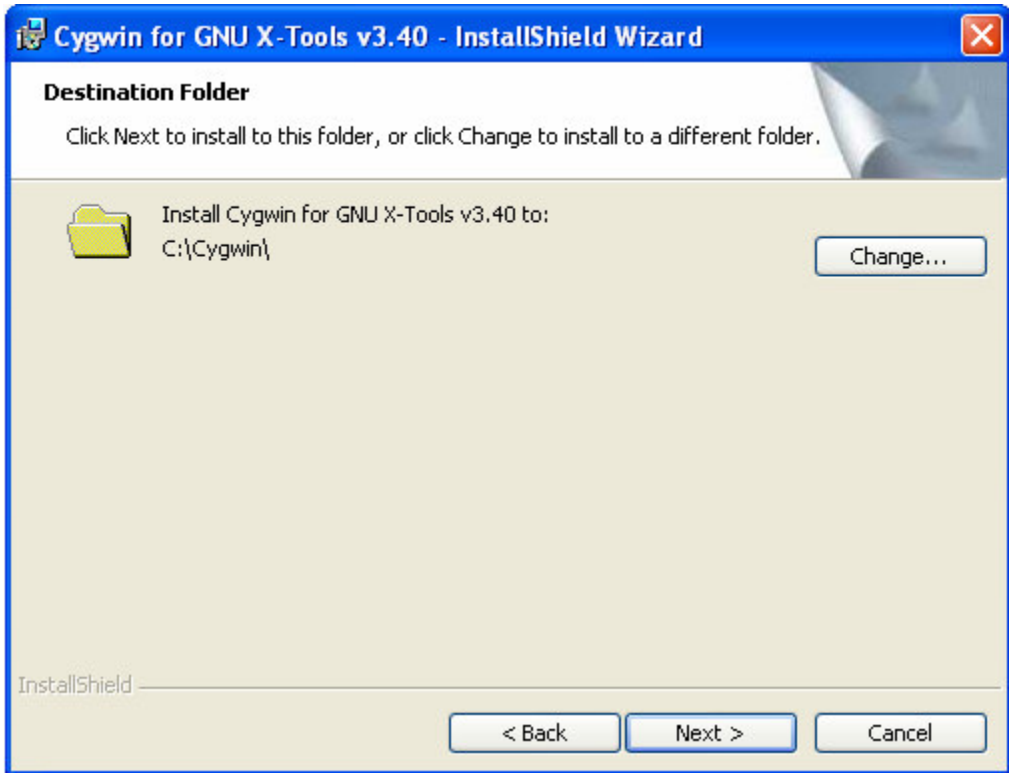
Click <Next> and the license agreement will appear.



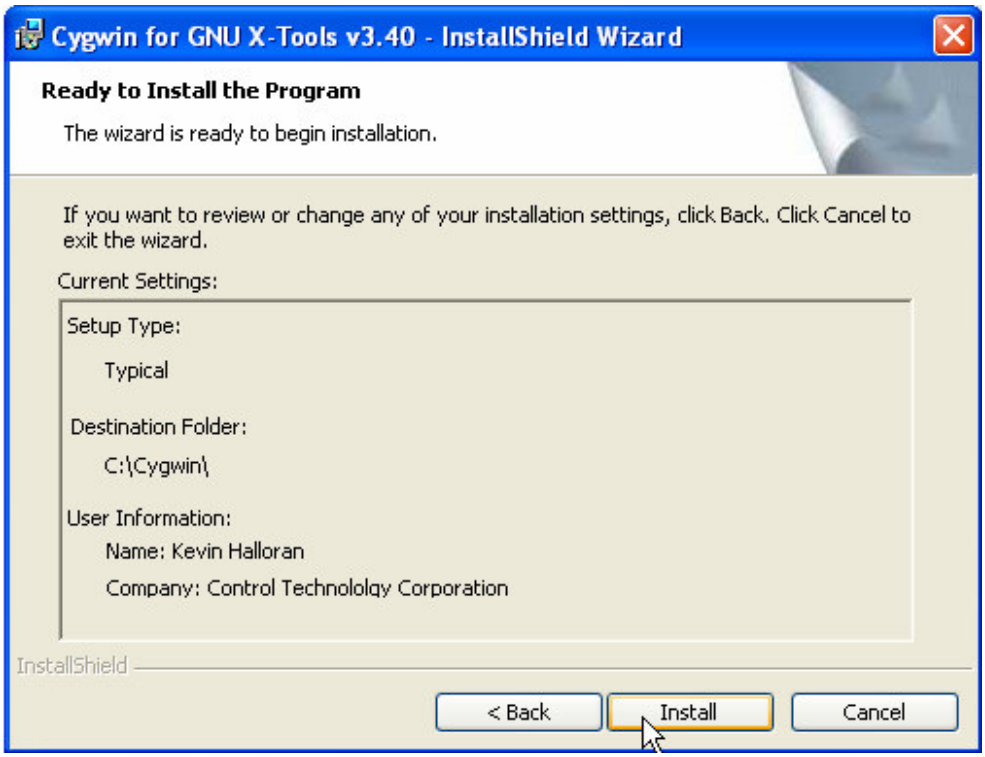
Click <I Accept> followed by <Next>. A screen requesting user information will appear.



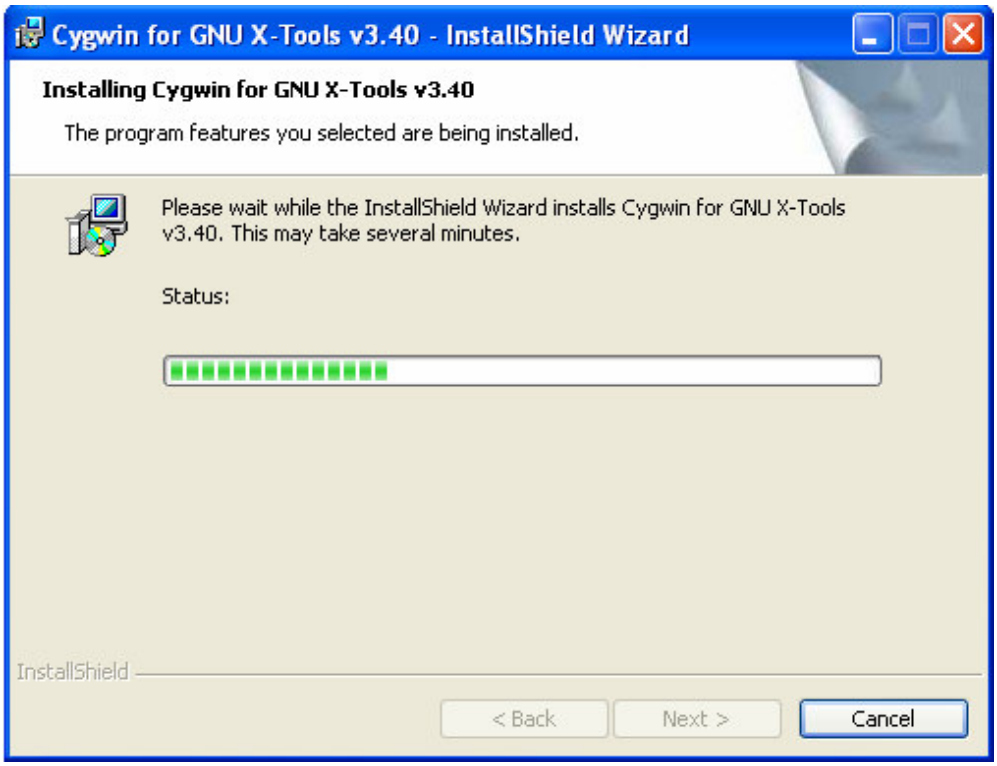
The default installation folder will appear, click <Change> if anything other than C:\Cygwin is desired, then click <Next> to proceed.



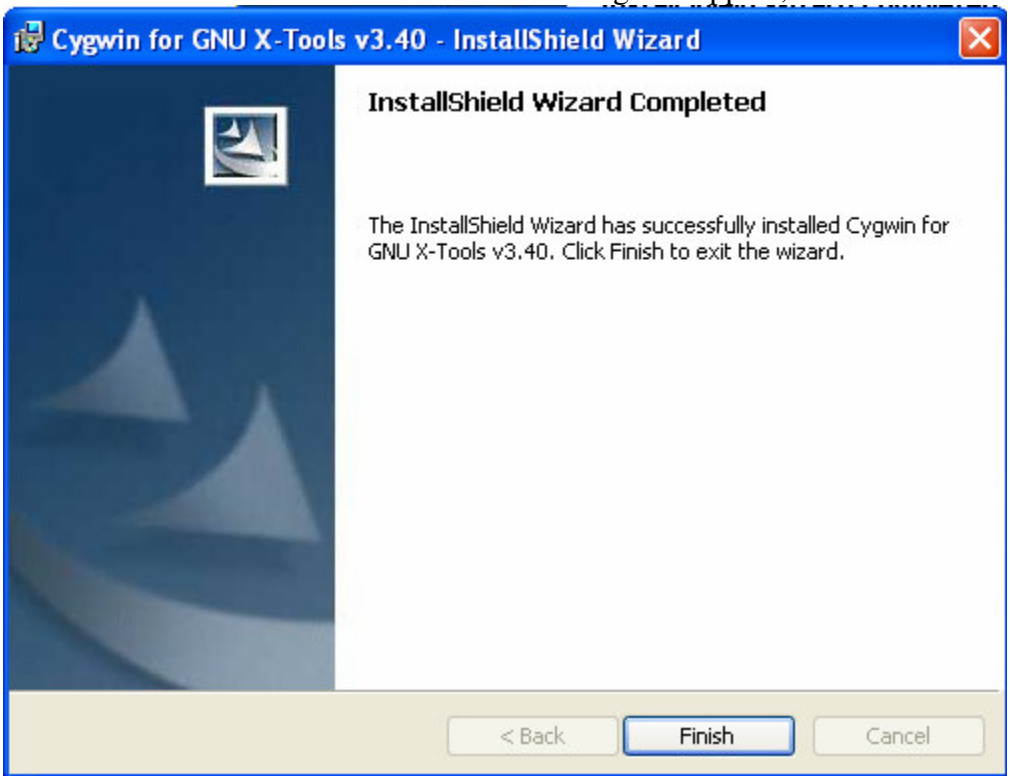
A summary of the current settings will appear, click <Back> to make changes or <Install> to begin:



Installation will begin and a progress bar and 'Status' message will appear:



Once the installation is finished the below dialog will appear, click <Finish>:



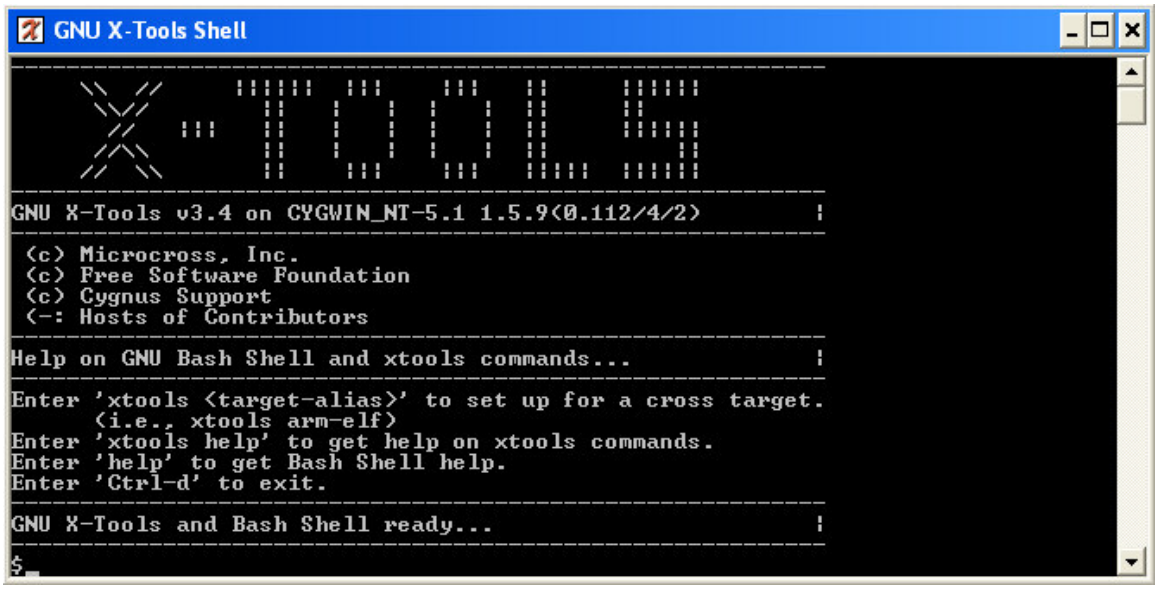
The generic ‘C’ compiler is now installed. In next section the CD containing the specific tools and libraries for the ARM environment will be installed.

GNU X-Tools V3.0b for Windows Installation

Cygwin v1.5.9 must be installed prior to proceeding.

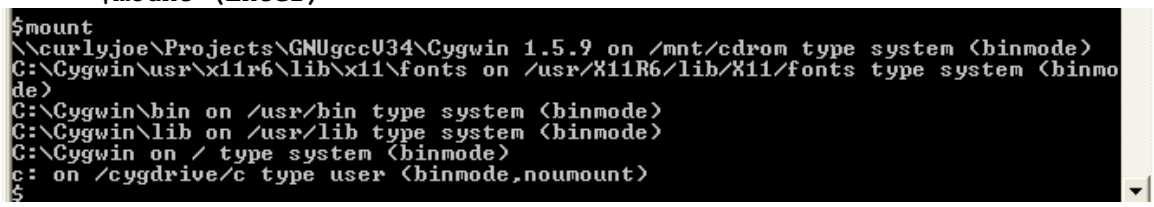
Start the **GNU X-Tools Shell** using the desktop icon that was installed with Cygwin.

You should get the GNU X-Tools banner upon startup of the command shell. This shell is for inputting UNIX style commands.



Insert the GNU X-Tools CD into your CD drive or unzip the appropriate file to a temporary directory. Type the following commands to install your tool-suite of choice:

```
$mount (Enter)
```



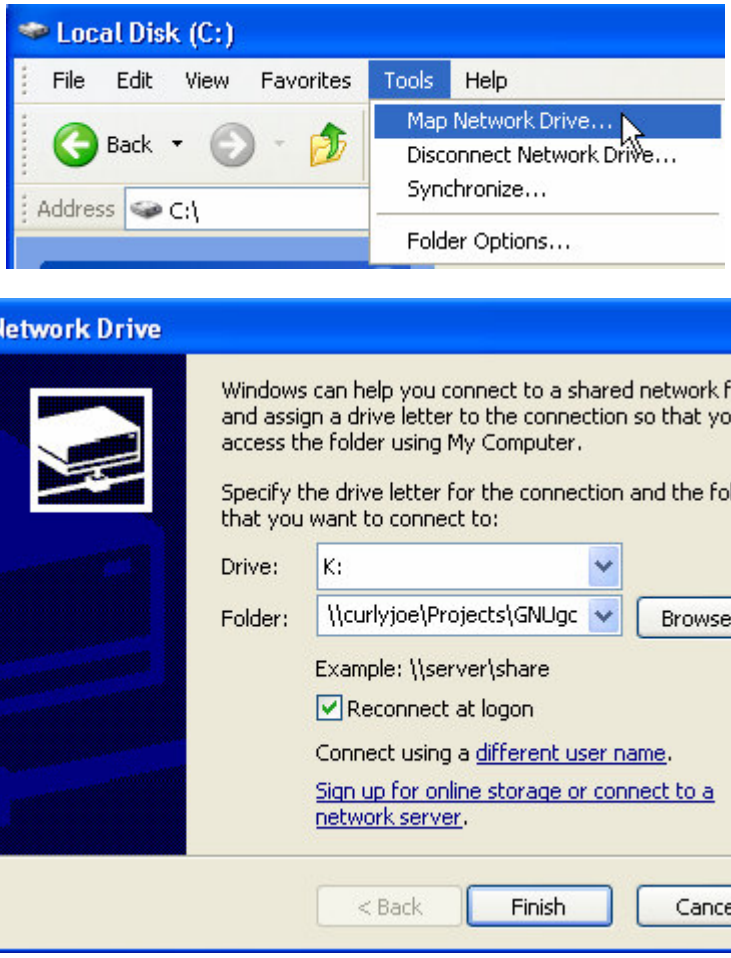
If the mount of the '/mnt/cdrom' is mapped to your CDROM drive letter with GNU X-Tools, then skip over to Installation of GNU X-Tools; otherwise, perform the following steps to mount your CDROM to the proper drive letter. From then on the mount will stay permanent unless changed:

```
$ umount -s /mnt/cdrom (Enter)
```

```
$ mount -s -b -f <CD-drive-letter>:  /mnt/cdrom  (Enter)
  e.g., mount -s -b -f  d:  /mnt/cdrom  (Enter) (assuming d: drive)
```

If you are referencing an area on a hard disk that the xtools were unzipped to then mount the cdrom pointing to the mapped disk drive, which is pointing to that directory.

Example: Unzipped files are in the 'Xtools 3.4' subdirectory on the \\curlyjoe\Projects\GNUgccV34\XTools 3.4 server drive. Therefore using Windows tools create a mapped drive, for instance K: pointing to this subdirectory:



Now mount the mapped drive:

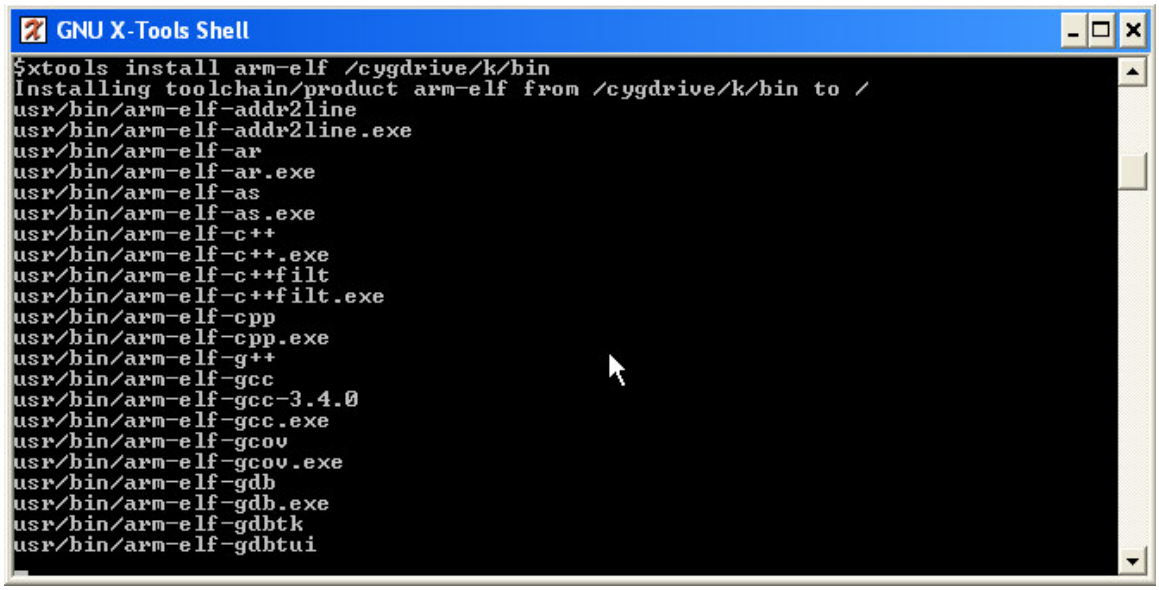
```
$ mount -s -b -f  K:  /cygdrive/k  (Enter) (assuming K: drive)
```

```
$mount
C:\Cygwin\usr\x11r6\lib\x11\fonts on /usr/X11R6/lib/X11/fonts type system (binmo
de)
C:\Cygwin\bin on /usr/bin type system (binmode)
C:\Cygwin\lib on /usr/lib type system (binmode)
C:\Cygwin on / type system (binmode)
c: on /cygdrive/c type user (binmode,noumount)
k: on /cygdrive/k type user (binmode,noumount)
$_
```

Typing the mount command, as above now shows the /cygdrive/k has been created and is mapped to the K: drive.

Now we are ready to install the X-Tools and Visual GDB for the ARM processor. First install the X-Tools and ARM specific libraries:

$ xtools install arm-elf /cygdrive/k/bin (Enter)



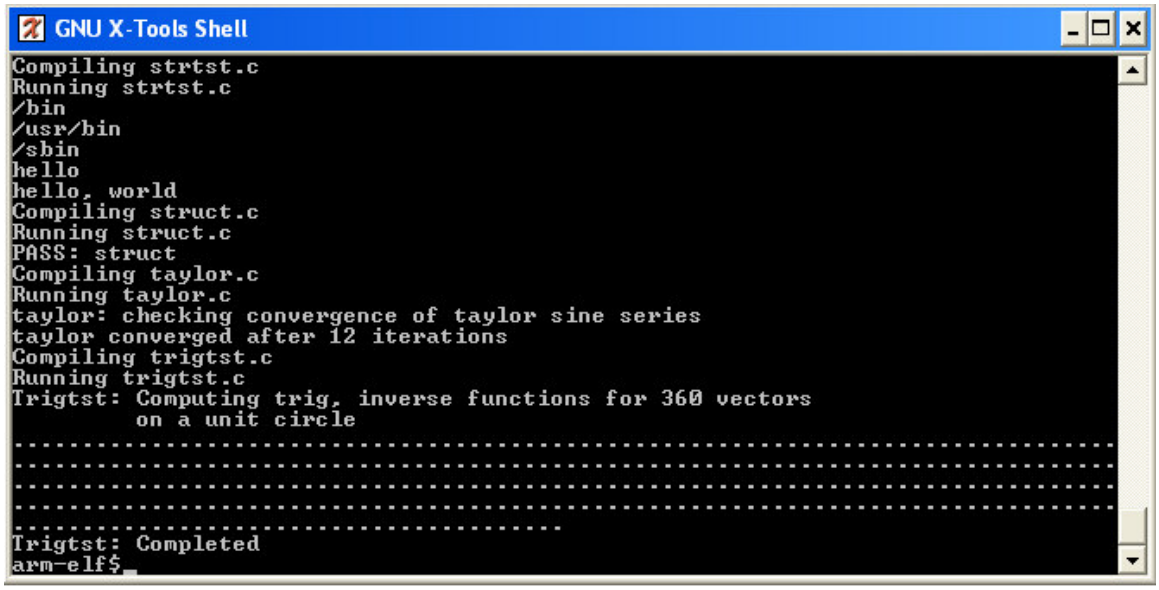
Verify Compiler Installation

Once installed, you can easily verify the proper operation of your toolsuite by running a test suite in the Cygwin directory. Follow these steps using the GNU X-Tools Shell:

$ xtools arm-elf (Enter)

arm-elf$ cd /home/test (Enter)

arm-elf$ ./run-all (Enter) (dot forward slash run-all)



Upon completion select <Exit> and reboot your PC.

Final System Configuration

To ensure a proper installation you should build the test program. Change your current directory to where you put the 5300 User C software and type build5300. Output should appear similar to below:

```
C:\5300UserC>build5300

C:\5300UserC>erase aload.sr1

C:\5300UserC>erase *.o

C:\5300UserC>erase aload.x

C:\5300UserC>erase makefile

C:\5300UserC>copy 5300\*.* .
5300\aload.sr1
5300\aload.x
5300\makefile
        3 file(s) copied.

C:\5300UserC>make
arm-elf-gcc -g -O -mcpu=arm9tdmi -Wall -fno-builtin -c -I./Headers -I. -I../libs
 -DLITTLE_ENDIAN -DMODEL5200   -c -o UserStart.o ./Source/UserStart.c
arm-elf-gcc -g -O -mcpu=arm9tdmi -Wall -fno-builtin -c -I./Headers -I. -I../libs
 -DLITTLE_ENDIAN -DMODEL5200   -c -o CoreFunc.o ./Source/CoreFunc.c
arm-elf-gcc -g -O -mcpu=arm9tdmi -Wall -fno-builtin -c -I./Headers -I. -I../libs
 -DLITTLE_ENDIAN -DMODEL5200   -c -o userApp.o ./Source/userApp.c
arm-elf-ld -o aload.abs UserStart.o CoreFunc.o userApp.o -Taload.x -Map aload.ma
p -L/usr/lib/gcc-lib/arm-elf/3.4.0/ -L../libs   -lstdc++ -lm -lc -lgcc
arm-elf-objcopy -R .comment -S -O srec aload.abs aload.sr
./specmem aload.sr aload.sr1
arm-elf-objcopy -R .comment -O elf32-little aload.abs aload.elf
Done
rm aload.sr

C:\5300UserC>
```

A file called *aload.sr1* should now appear. This file is the loadable module to be executed by the controller. Its file size should be about 118,880 bytes. This sample program simply takes any value written to registers 2 through 6 and divides it by 2. It also contains an example on how to use the serial and UDP network ports. Its source code is contained within UserApp.c.

As a test install this file on your controllers /_system/Programs subdirectory and then by using telnet execute "run userprogram aload.sr1". Write a 10 to register 2 and then read it back, the result should be a 5.

Setting up the 'C' programming environment for your particular editor or specially configuring directory structures is beyond the scope of this document. Refer to the documentation supplied by Microcross when necessary. Typically source files are placed in the *Source* subdirectory and header files in the *Header* subdirectory. If a new source file is added make sure to also edit the makefile and add where needed, using an existing file as an example.

CHAPTER

3

Loadable 'C' User Functions and Filters

'C' User Functions are compiled and linked into a single .sr1 file. This file is placed in the /_system/Programs directory, on the flash disk. It is loaded using the "run userprogram filename.sr1" script command, into SDRAM memory (reference map). The 'run' script command can either be executed at the command line, via Telnet, or embedded in a script file and executed by writing the script number to register 12311. Reference the Script Language Guide; #951-520003 (5200), for additional information.

5300 System Memory Map

FLASH

Location	*Address Range*	*Max Bytes*	*Bus*
Main Memory Board – Boot Monitor and tfs/elf image storage for booting main program	0x10000000 to 0x103FFFFF Not available for user. (320 series flash chip)	4M	16
Main Memory Board – Flash Disk	0x14000000 to 0x17FFFFFF (up to 320 series flash)	4M	16
Main Memory Board – Flash Disk (optional)	0x01800000 to 0x01BFFFFF (up to 320 series flash)	4M	16
Total Maximum		**12M**	

NV-RAM (Battery Backed)

Location	*Address Range*	*Max Bytes*	*Bus Width*
Main Memory Board – Static Ram	0x80000000 to 0x801FFFFF	2M (first 1M reserved*) *subject to change	32

Top Memory Expansion Board	0x80200000 to 0x802FFFFF	2M	32
Total Maximum		**4M (3M user)**	

SDRAM (Dynamic Memory, volatile at hardware reset and power down)

Location	*Address Range*	*Max Bytes*	*Bus Width*
Main Board – Execution area for firmware and 'C' User programs	0x20000000 to 0x21FFFFFF 0x20000000 to 0x20FFFFFF (16M) Program execution area, copied from flash, as well as heap space (about 11M) 0x21000000 to 0x215FFFFF (6M) User available area, volatile ram disk, etc. CTC reserves the right to use for feature enhancements thus if use start with high memory first. 0x21600000 to 0x21BFFFFF (6M) 'C' Development area 0x21C00000 to 0x21FFFFFF (4M) Available, FTP re-flash area @ 0x06c00000.	32M (12M User)	32
Total Maximum		**16M** **(12M User)**	

Resource Filter Example

A Resource Filter allows a 'C' User Function to modify a value prior to the application program receiving it or on a write operation, prior to it being written to the actual resource. To implement a filter a User Function must first be registered with the Quickstep OS, along with what Resource parameters will cause it to be invoked. A RESOURCE_INFO structure is filled out and passed to the "addResourceFilter" function, specifying the type of resource to be monitored, read and/or write operation, and the assigned resource number range. For example, adding a filter to register #2 is given below. Only register 2 is monitored since the start and end range are the same:

```
RESOURCE_INFO resource;
void *handle;

   // Lets install a filter function as a sample
   resource.type = RESOURCE_REGISTER;
   resource.start = 2;              // lets filter register 2
   resource.end = 2;                // no range, only 2 for now
   resource.mode = RESOURCE_READ;   // read operation only
```

```
// Now add the filter...
handle = addResourceFilter(&resource,sampleFilter);
```

The sampleFilter function will simply divide any read operations by 2. Therefore, if Quickstep or CTCMON were to read register 2 and a 40 was contained in it, the value actually read back would be 20. This same technique can be used for any available resource. By changing `resource.end` to a 6 (as in supplied sample program `UserApp.c`), a range of registers can be specified. Below is a simple filter function:

```
int sampleFilter(void *handle, FILTERPARAMS *params, STDVAL value, RETVAL *status)
{
        // This sample filter simply processes a read or write operation on a register
        switch(params->mode)
        {
                case RESOURCE_READ:
                        // Someone is attempting to read the register, actual value in "value"
                        // Let's divide it by 2 just for test purposes
                        if (value)          // don't divide by 0...
                                value = value/2;
                        break;
                case RESOURCE_WRITE:
                        // This won't be called since we only defined for READ, bits are Or'ed
                        break;
        }
        // Return new or same value to use
        return value;
}
```

When finished with the filter, during cleanup, make sure you call `releaseResourceFilter`, passing the handle of the resource.

Virtual Table, User and Quickstep

It is recommended that the programmer reference the sample programs included in the distribution. Much of the detail provided below references the internal operations and is not necessarily needed to construct user functions.

Two virtual tables exist which expose functions that are available, both of which are defined in `uTable.h`. VTABLE_QS is the table provided by the 5300 OS and contains the function calls available for the user program (other than the standard C library). VTABLE_USER is the table that is filled in by the user program, as required. As a quick reference the tables are defined below:

```
/*************************************************************
*  struct vtableQS - Quickstep OS Function table             *
*                                                            *
*  Virtual function table for access to Quickstep functions. *
*  This table is initialized by Quickstep OS at powerup.     *
*************************************************************/
typedef struct vtableQS
{
        REGISTER_GET        regRead;                    // read a Quickstep register
        REGISTER_PUT        regWrite;                   // write a Quickstep register

        COMM_SENDMSG        commSendMsg;                // Send a string out a serial port
        COMM_GENERICCMD     commGenericCmd;             // Send a command to a serial port
        COMM_GET_RQST       commReadMsg;                // Read a message from a serial port

        ADD_RESOURCE_FILTER     addResourceFilter;      // add a resource filter to the access list
        REMOVE_RESOURCE_FILTER  removeResourceFilter;   // remove a resource filter from the access list

        GET_SYSTEM_TICS     systemTics;         // Returns number of tics since powerup, in ms.

        PRINTF              printf;             // printf function redirected to UDP debugger output screen
        SPRINTF             sprintf;            // sprintf function used by Quickstep, protected by mutex
        LOGEVENTPLUS        logEventPlus;       // Log a predetermined value type and value/string to the event log
        LOGEVENT            logEvent;           // Log a low level value into the debug event log
        FIREWATCHDOG    fireWatchdog;           // Reset a watchdog timer, Quickstep calls it continually.

        // If for any reason not return control must be called in

        // less than 86 ms. or reset will occur.

        ENTER_STEP_ATOMICITY    enterStepAtomicity;     // Obtain ownership of step atomicity
        EXIT_STEP_ATOMICITY     exitStepAtomicity;      // Release ownership of step atomicity...

        /*************************************************************/
        // below virtual functions are for advanced use only and not
        // offered for general support
        /*************************************************************/
        MOTION_GET_NUMMOTORS    Motion_GetNumMotors;    // Get number of motion objects in system
        MOTION_GET_ATTRIBUTE    Motion_Get_Attribute;
        MOTION_PUT_ATTRIBUTE    Motion_Put_Attribute;
        MOTION_READY        Motion_Ready;               // Get state of motion object
        MOTION_SIMPLE_COMMAND   Motion_Simple_Command;

        THREAD_CREATE       thread_create;              // Create a new thread
        THREAD_SUSPEND      thread_suspend;             // Suspend an existing thread
        THREAD_RESUME       thread_resume;              // Allow a suspended thread to resume operation
        THREAD_SLEEP        thread_sleep;               // Sleep for the specified number of tics (1ms/tic)
        MUTEX_CREATE        mutex_create;               // Create a mutex object
        MUTEX_GET           mutex_get;                  // get control of the mutex
```

```
        MUTEX_PUT           mutex_put;              // give control of the mutex back
        MUTEX_DELETE        mutex_delete;           // Destroy the mutex, giving all memory back

        BYTE_ALLOCATE       tx_byte_allocate;       // NOT TO BE USED EXCEPT FOR INTERNAL ROUTINES
        BYTE_RELEASE        tx_byte_release;        // NOT TO BE USED EXCEPT FOR INTERNAL ROUTINES
        BYTE_POOL_CREATE    tx_byte_pool_create;    // NOT TO BE USED EXCEPT FOR INTERNAL ROUTINES
        BYTE_POOL_DELETE    tx_byte_pool_delete;    // NOT TO BE USED EXCEPT FOR INTERNAL ROUTINES
        INTERRUPT_CONTROL   interruptControl;       // NOT TO BE USED EXCEPT FOR INTERNAL ROUTINES

        COMM_NETWORKOPEN    commNetworkOpen;  // Open a network connection/socket spawning RX thread
        COMM_NETWORKCLOSE   commNetworkClose;       // Close a network connection
        COMM_NETWORKSEND    commNetworkSend;        // Send data on a network connection
/********** BELOW ONLY COMPATIBLE WITH 5300 CONTROLLER ************/
        REGISTER_VGET       regVRead;               // read a Quickstep register as a Variant
        REGISTER_VPUT       regVWrite;              // write a Quickstep register as a Variant

        REGISTER_LOCK   regLock;                    // Register mutex lock set
        REGISTER_UNLOCK     regUnlock;              // Register mutex unlock

        TASK_SETBRANCH      taskSetBranch;          // Set the step to branch to upon return from 'C' API
        TASK_STEPLOCK       taskStepLock;           // Set the step lock flag to force execution of next step

        // of present task without loosing atomicity
        TASK_CREATETASK     createTask;             // Create a Quickstep Task
        TASK_KILLTASK       killTask;               // Kill a Quickstep Task
        TASK_GETTASKFROMHANDLE      getTaskFromHandle;      // Get TASK * from a TASK handle
        RUNCMD_ALLOCATE  allocateCommandParser;     // Allocate an environment for Script Command Processing
        RUNCMD_RELEASE  releaseCommandParser;       // Free an environment used for Script Command Processing
        RUNCMD_RUN          runCommandParser;   // Run a Script Command
        RUNCMD_GETRESPONSE  getCommandResponse;  // Get a Script Command response, copying it to a local buffer

        // File system calls

        CTC_FOPEN       CTC_fopen;   // Open the desired file to type of access desired and return handle for further access
        CTC_FILELENGTH  CTC_filelength;     // Get file length
        CTC_FCLOSE      CTC_fclose;         // Close file openned by CTC_fclose()
        CTC_FEOF        CTC_feof;           // Check for end of file
        CTC_FWRITE      CTC_fwrite;         // Write to file
        CTC_FTELL       CTC_ftell;          // Get current file position
        CTC_FFLUSH      CTC_fflush;         // Flush open file
        CTC_FREAD       CTC_fread;          // Read from file
        CTC_FGETS       CTC_fgets;          // Get a line terminated with CR LF from an open file
        CTC_FPUTS       CTC_fputs;          // Write a null terminated string to an open file
        CTC_FSEEK       CTC_fseek;  // Seek to specific position in a file.  Origin is ignored, always from start of file
        CTC_GETFILENAME   CTC_getFileName;  // Return pointer to the name of the openned file within a directory block

        // DIRECTORY FUNCTIONS

        CTC_RMDIR       CTC_rmdir;          // Delete a directory
        CTC_PWD         CTC_pwd;            // Get current position in a directory tree
        CTC_CD          CTC_cd;             // Change directories
        CTC_MKDIR       CTC_mkdir;          // Create a new directory
        CTC_FORMATDISK CTC_formatDisk;      // Format the disk
        CTC_GETFULLFILENAME     CTC_getFullFileName;   // Get full file name with path information
        CTC_VOLUMEGETINFO       CTC_volumeGetInfo;// Get Flash disk information at disk level
        CTC_FINDFIRST   CTC_findfirst;  // Initialize CTC_FIND_DATA structure and get information on first file
        CTC_FINDNEXT    CTC_findnext;       // Get CTC_FIND_DATA information on next file in directory tree
        CTC_FINDCLOSE   CTC_findclose;      // Terminate the directory tree walk initiated with the CTC_findfirst() call
        CTC_REMOVE      CTC_remove;         // Delete a file
        CTC_RENDIR      CTC_rendir;         // Rename a directory (ram disk and SDISK only)
        CTC_RENAME      CTC_rename;         // Rename a file
        CTC_GETFILEINFO CTC_getFileInfo;    // Get information on a file that is not open, by name and path only

} VTABLE_QS;


/*************************************************************
*  USER VIRTUAL FUNCTION TABLE PROTOTYPE DEFINITIONS         *
*                                                            *
*  Virtual function table hooks for access by Quickstep OS   *
```

```
*  to call if defined during strategic points of system                    *
*  execution.                                                              *
*  This table is initialized by user within their main()                   *
*  function, with their 'C' functions, as needed, else place               *
*  a NULL in the table entry                                               *
**************************************************************************/
typedef int   (*USER_INITIALIZATION)(void);       // Called when started, = main
typedef int   (*USER_OPERATION)(void *);          // Called to invoke a user operation

typedef void  (*TIMER_INTERRUPT_HOOK)();      // invoked every timer tic, don't stay here long!!!
typedef int   (*TASK_LOOP_HOOK)(SYSMODE state);          // invoked after all Quickstep tasks have executed, round robin
typedef    int   (*SUPERTASK_HOOK)(SYSMODE state);        // invoked after each Quickstep task
typedef    int   (*SERVICELOOP_HOOK)(SYSMODE state);   // invoked after each Quickstep task
typedef void  (*END_ALL_TASKS_HOOK)();                  // invoked if all tasks are told to end.
typedef void  (*USER_SHUTDOWN)();                       // invoked if about to load another user module on top of this one.
typedef void  (*NEWTASKSTARTED)(TASK *task);            // invoked when a new task is beginning, created.
typedef void  (*TASKENDED)(TASK *task);                 // invoked when a task is shutting down
typedef void  (*QS4CONTROL)(int reason, void *info);    // invoked periodically with 'reason' and 'info', reason specific.
typedef void  *(*QS4OBJECTTABLE)(int *size);            // invoked to access QS4 object table pointer and size
typedef int (*QS4_OBJACCESS)(int write, int objnum, int propnum, int proptype, void *resultVal);  // invoked to access QS4 object
                                                        // property value when indirection flag set

/*************************************************************
*  struct vtableUser - User Defined Function table           *
*                                                            *
*  Virtual function table for access to User functions.      *
*  This table is initialized by user within their main()     *
*  function, upon initial loading into memory and prior to   *
*  operation                                                 *
*************************************************************/
#define VALID_USER_SIGNATURE  0x10adf7L      // Special identifier at start of User table, required
#define USER_VERSION_MASK     0xffff0000L
#define USER_ENTRIES_MASK     0x000000ffL

#define USER_VERSION_REQUIRED 0x00010000L // Not currently used
#define USER_ENTRIES_REQUIRED 13          // Number of function entries in the vtableUser structure
typedef struct vtableUser
{
        long              signature;          // Unique value to verify table is correct type
        long              version;            // Version control information to ensure compatibility
        USER_INITIALIZATION   initialize;     // User routine to be called at start, main() function
        USER_OPERATION        function;       // NOT USED
        TIMER_INTERRUPT_HOOK  timerTic;       // Invoked on every timer tic (1ms) if defined
        TASK_LOOP_HOOK        taskLoop;       // Invoked as a normal Quickstep step if defined
        SUPERTASK_HOOK        superTaskLoop;  // Invoked after each Quickstep step, like a super task
        SERVICELOOP_HOOK      serviceLoopHook;  // Invoked outside the Quickstep main loop when steps not running
        END_ALL_TASKS_HOOK    endAllTasks;    // Invoked if all tasks are being shutdown by a Quickstep Cancel command
        USER_SHUTDOWN  userShutdown;          // Invoked when a new "C" module is being loaded for cleanup purposes
        NEWTASKSTARTED  taskStart;            // Invoked when a new task is beginning, created.
        TASKENDED             taskEnd;        // invoked when a task is shutting down
        QS4CONTROL          qs4Control;       // invoked periodically, based on specific 'reason' parameter
        QS4OBJECTTABLE   QS4ObjectTable;      // invoked to access QS4 object table pointer and size.
        QS4_OBJACCESS      QS4ObjAccess;      // invoked to access QS4 object table property value when indirection set)
} VTABLE_USER;
```

The function definitions and calling parameters are discussed in the "VTABLE_QS Function Prototypes and Definitions" section. The VTABLE_USER contains entries which are previously set up and may be modified by the user. The table exists at the end of the `CoreFunc.c` file:

```
// Below is main table referenced by the Quickstep Operating System, tread carefully when changing
// as it must match that of the OS...  NOTE:  Change version and entries number in table!!!
const VTABLE_USER localUserCommand __attribute__ ((section (".ftable"))) = {
   VALID_USER_SIGNATURE,      // signature
   0x0001000C,     // version 00.01, 12 entries
   main,              // initialize
```

```
    NULL,           // function unused
    NULL,           // timerTic;
    taskLoop,       // taskLoop;
    NULL,           // superTaskLoop;
    NULL,           // serviceLoopHook;
    NULL,           // endAllTasks;
    userShutdown,   // userShutdown;
    taskStart,  // newTaskStarted;  (called if non-null when task first created, allocate task user memory)
    taskEnd,    // taskEnded;  (called when a task is ending, free user task memory
   NULL,        // pidControl; invoked periodically, after all analog inputs are scanned
   NULL         // QS4ObjectTable;          invoked to access QS4 object table pointer and size.
};
```

Twelve 5300 OS hooks are available for user modification. If a NULL is present then no operation will be performed for that particular function call.

main – Required 'C' user Program entry point upon being loaded. Any initialization should be done here and control promptly returned to the calling functions.

timerTic – This function will be called once per 5200 timer tic, from the interrupt level, approximately 1 millisecond/tic. Control must be returned to the 5200 immediately.

taskLoop – This function will be called once per Quickstep step loop, just like any other task. Quickstep steps are executed round-robin. Upon return the first Quickstep task will have a step executed since the 'C' user function taskLoop is always the last task called. Note that step atomicity is maintained.

superTaskLoop – Same as taskLoop, except this function is called after each Quickstep task executes a step.

endAllTasks – This function is called to notify the user function that there has been a task "Cancel" command executed by Quickstep, causing task execution to stop, and that any cleanup that should be done needs to be done now.

userShutdown – This function is provided in `CoreFunc.c` and is called whenever a user program is about to be unloaded from memory and a new one loaded. Frees all resources that have been allocated and returns control.

taskStart – This function is provided in *CoreFunc.c* and is called whenever a Quickstep task is first started. It allows the user program to allocate memory or prepare for anything task specific. Reserved for QuickBuilder.

taskEnd – This function is provided in *CoreFunc.c* and is called whenever a Quickste task finishes, is done executing. Reserved for QuickBuilder.

qs4Control – This function is provided in *CoreFunc.c* and is called periodically with analog values based upon scan rate. Used for PID control and reserved for QuickBuilder.

QS4ObjectTable – QuickBuilder use only.

QS4ObjAccess – QuickBuilder use only.

VTABLE_QS Function Prototypes and Definitions

Below are a few multipurpose functions which were available on the 5100/5200. Reference 'CoreFunc.c' and the "Model 5300 Enhancement Overview", document 951-530001, for additional available functions, including full file system access.

```
/*****************************************************************************/
/*                                                                           */
/*  FUNCTION                                                  RELEASE        */
/*                                                                           */
/*    main                                                  PORTABLE C       */
/*                                                             1.0           */
/*  DESCRIPTION                                                              */
/*                                                                           */
/*    This function is the main input function called after a User           */
/*    C file is loaded into memory for execution.  Any initialization        */
/*    required should be done and control returned.                          */
/*                                                                           */
/*    MAKE SURE TO RETURN CONTROL AND DON"T TAKE LONG!!!  IF CALLED BY       */
/*    A QUICKSTEP RUNNING A SCRIPT, YOU MAY HAVE TO INVOKE A WATCHDOG        */
/*    RESET FUNCTION IF SPEND MORE THAN 40 MS HERE  (fireWatchdog())         */
/*                                                                           */
/*  INPUT                                                                    */
/*                                                                           */
/*    none                                                                   */
/*                                                                           */
/*  OUTPUT                                                                   */
/*                                                                           */
/*    0 = Initialization successful, allow User functions to execute         */
/*    non-zero = Init failed, do not run User functions                      */
```

```
/*                                                                          */
/*  CALLS                                                                   */
/*    - __main() call inserted by the compiler prior to any other code      */
/*    - user define initialization routines, as required                    */
/*                                                                          */
/*  CALLED BY                                                               */
/*                                                                          */
/*    C User Function file loader (Quickstep OS)                            */
/*                                                                          */
/*  RELEASE HISTORY                                                         */
/*                                                                          */
/*    DATE              NAME                        DESCRIPTION             */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
int main (void);
```

Register Access

```
/****************************************************************************/
/*                                                                          */
/*  FUNCTION                                              RELEASE           */
/*                                                                          */
/*    regRead                                           PORTABLE C          */
/*                                                         1.0              */
/*  DESCRIPTION                                                             */
/*    Read a Quickstep Register value.                                      */
/*                                                                          */
/*  INPUT                                                                   */
/*    UINT16 RegNum - Register number from 1 to 64535 to read               */
/*    INT32 *RegVal - Pointer to a 32 bit wide Integer to store the result  */
/*                                                                          */
/*  OUTPUT                                                                  */
/*    RETVAL -                                                              */
/*      SUCCESS = function called properly                                  */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                    */
/*      (also register specific return values defined in Errors.h)          */
/*                                                                          */
/*  CALLS                                                                   */
/*    Quickstep OS virtual table function pointer                           */
/*                                                                          */
/*  CALLED BY                                                               */
/*    As required by user code                                              */
/*                                                                          */
/*  RELEASE HISTORY                                                         */
/*                                                                          */
/*    DATE              NAME                        DESCRIPTION             */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
RETVAL regRead(UINT16 RegNum, INT32 *RegVal);
```

```
/*****************************************************************************/
/*                                                                           */
/*  FUNCTION                                                  RELEASE        */
/*                                                                           */
/*    regWrite                                               PORTABLE C      */
/*                                                              1.0          */
/*  DESCRIPTION                                                              */
/*    Write a value to a Quickstep Register                                  */
/*                                                                           */
/*  INPUT                                                                    */
/*    UINT16 RegNum - Register number from 1 to 64535 to read                */
/*    INT32  RegVal - 32 bit wide Integer value to store                     */
/*                                                                           */
/*  OUTPUT                                                                   */
/*    RETVAL -                                                               */
/*      SUCCESS = function called properly                                   */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                     */
/*      (also register specific return values defined in Errors.h)           */
/*                                                                           */
/*  CALLS                                                                    */
/*    Quickstep OS virtual table function pointer                            */
/*                                                                           */
/*  CALLED BY                                                                */
/*    As required by user code                                               */
/*                                                                           */
/*  RELEASE HISTORY                                                          */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
RETVAL regWrite(UINT16 RegNum, INT32 RegVal);
```

Communications

```
/*****************************************************************************/
/*                                                                           */
/*  FUNCTION                                                  RELEASE        */
/*                                                                           */
/*    commSendMsg                                            PORTABLE C      */
/*                                                              1.0          */
/*  DESCRIPTION                                                              */
/*    Send a buffer of characters out a serial communications port           */
/*                                                                           */
/*  INPUT                                                                    */
/*    INDX port - Serial port to send buffer out on, 1 is COM1, 2 is COM2,   */
/*                virtual TCP connections are 3 to 7                         */
/*    UINT8 *msg - Pointer to unsigned character buffer containing message   */
/*    UINT16 size - Length of message to send                                */
/*                                                                           */
/*  OUTPUT                                                                   */
/*    RETVAL -                                                               */
/*      SUCCESS = function called properly                                   */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                     */
/*      (also register specific return values defined in Errors.h)           */
/*                                                                           */
/*  CALLS                                                                    */
```

```
/*    Quickstep OS virtual table function pointer                          */
/*                                                                         */
/*  CALLED BY                                                              */
/*    As required by user code                                             */
/*                                                                         */
/*  RELEASE HISTORY                                                        */
/*                                                                         */
/*    DATE              NAME                        DESCRIPTION            */
/*                                                                         */
/*                                                                         */
/***************************************************************************/
RETVAL commSendMsg( INDX port, UINT8 *msg, UINT16 size);

/***************************************************************************/
/*                                                                         */
/*  FUNCTION                                                 RELEASE       */
/*                                                                         */
/*    commGenericCmd                                        PORTABLE C     */
/*                                                             1.0         */
/*  DESCRIPTION                                                            */
/*     This function sends a generic message command (undefined format) of*/
/*     length "size" to the specified "port" using the appropriate driver.*/
/*     Any response is copied back to "msg"; the size of the of the        */
/*     response it copies back to "size".                                  */
/*     Messages can be used to change baud rate, etc...                    */
/*                                                                         */
/*  INPUT                                                                  */
/*    INDX port - Serial port to send buffer out on, 1 is COM1, 2 is COM2,*/
/*                virtual TCP connections are 3 to 7                       */
/*    UINT8 *msg - Pointer to unsigned character buffer containing message*/
/*    UINT16 *size - Pointer to unsigned short to store result length in  */
/*                                                                         */
/*  OUTPUT                                                                 */
/*    RETVAL -                                                             */
/*      SUCCESS = function called properly                                 */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                   */
/*      (also register specific return values defined in Errors.h)         */
/*                                                                         */
/*  CALLS                                                                  */
/*    Quickstep OS virtual table function pointer                          */
/*                                                                         */
/*  CALLED BY                                                              */
/*    As required by user code                                             */
/*                                                                         */
/*  RELEASE HISTORY                                                        */
/*                                                                         */
/*    DATE              NAME                        DESCRIPTION            */
/*                                                                         */
/*                                                                         */
/***************************************************************************/
/*  Below are msg[] contents commands for each available.  The command
must*/
/*  be the first byte of msg[] upon calling the function.
COMMCMD_RQST_QUERY - READ
      Check if the transmitter is free for message sending.
      Upon return:
```

```
      msg[0] = 0x00 if free.
      msg[1] = 0x01 if busy.
      *size = 1.
COMMCMD_RQST_CLRBUF - WRITE
      Clear the communications receive buffer.
      Upon return:
      msg[0] = COMMCMD_RSPN_ACK;
      *size = 1;
COMMCMD_RQST_PARSING - WRITE
      Turn parsing on or off based on the third byte of the message.
      Set msg to:
      msg[2] = 0 then disable parsing
      msg[2] = 1 then enable parsing
      Upon return:
      msg[0] = COMMCMD_RSPN_ACK;
      *size = 1;
COMMCMD_RQST_GETCNT - READ
      Return the current receive buffer count.
      Upon return:
      msg[0] = current count
      *size = 1;
COMMCMD_RQST_GETCH - READ
      Retrieve the nth character in the receive buffer.
      Set msg to:
      msg[2] = offset in buffer with 0 being first character
      Upon return:
      msg[0] = character at position requested
      *size = 1;
COMMCMD_RQST_SET_MODBUS - WRITE
      Activate/Deactivate Serial port modbus and set global port to use.
      Set msg to:
      msg[2] = Modbus RTU serial port address 1 to 254, also enables, 0
disables
      Upon return:
      msg[0] = COMMCMD_RSPN_ACK;
      *size = 1;
COMMCMD_RQST_NEWBAUD - WRITE
      Change the baud rate on the serial port (only physical, not virtual
work)
      msg[2] = baud rate desired where 1 = 600, 2 = 1200, 3 = 2400, 4 =
4800,
                5 = 9600, 6 = 19200 (default at powerup), 7 = 38400.
      Upon return:
      msg[0] = COMMCMD_RSPN_NACK if bad value, or COMMCMD_RSPN_ACK if OK
      *size = 1;
*/
RETVAL commGenericCmd( INDX port, UINT8 *msg, UINT16 *size);

/*************************************************************************/
/*                                                                       */
/*  FUNCTION                                              RELEASE        */
/*                                                                       */
/*    commReadMsg                                        PORTABLE C      */
/*                                                          1.0          */
/*  DESCRIPTION                                                          */
/*     This function invokes the driver associated with "port" and       */
```

```
/*     returns a pointer to the buffer that contains the request message. */
/*     The message contents are NOT copied.  If a request message from    */
/*     the port is not available, the "buf" pointer is set to NULL.  The  */
/*     message contents are assumed to be encoded in one of the standard  */
/*     protocols.  Size is set to the number of received bytes in the     */
/*     buffer, not the number of bytes in the message.                    */
/*                                                                        */
/*  INPUT                                                                 */
/*    INDX port - Serial port to send buffer out on, 1 is COM1, 2 is COM2,*/
/*                virtual TCP connections are 3 to 7                      */
/*    UINT8 **buf - Pointer to unsigned character buffer stored here or   */
/*                  NULL if no message                                    */
/*    UINT16 *size - Number of bytes in buffer                            */
/*  OUTPUT                                                                */
/*    RETVAL -                                                            */
/*      SUCCESS = function called properly                                */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                  */
/*      (also register specific return values defined in Errors.h)        */
/*                                                                        */
/*  CALLS                                                                 */
/*    Quickstep OS virtual table function pointer                         */
/*                                                                        */
/*  CALLED BY                                                             */
/*    As required by user code                                            */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                        DESCRIPTION           */
/*                                                                        */
/*                                                                        */
/**************************************************************************/
RETVAL commReadMsg( INDX port, UINT8 **buf,UINT16 *size);
```

Diagnostics

```
RETVAL logEvent(EventCode event, long parameter);
```

Resource Filters

```
/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                                RELEASE       */
/*                                                                        */
/*    addResourceFilter                                    PORTABLE C     */
/*                                                           1.0          */
/*  DESCRIPTION                                                           */
/*    This function is called to add a resource filter to the Quickstep   */
/*    list.  The resource to monitor is defined in the RESOURCE_INFO      */
/*    structure and passed as a parameter.  A pointer to the 'C' function */
/*    to call upon access is also passed                                  */
/*                                                                        */
/*  INPUT                                                                 */
/*    RESOURCE_INFO *rsc - Pointer to structure defining resource and     */
/*       access method upon which to invoke the passed function           */
```

```
/*    FILTER_FUNCTION func - Pointer to 'C' function to call when the     */
/*    parameters of the RESOURCE_INFO structure are satisfied             */
/*                                                                        */
/*  OUTPUT                                                                */
/*    void *handle - Handle returned by addResourceFilter or NULL if      */
/*                   failed                                               */
/*                                                                        */
/*  CALLS                                                                 */
/*    Quickstep OS virtual table function pointer                         */
/*                                                                        */
/*  CALLED BY                                                             */
/*    As required by user code                                            */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
/*                                                                        */
/*                                                                        */
/**************************************************************
*  RESOURCE_INFO rsc - addResourceFilter parameter block      *
*                                                             *
*  This parameter block must be filled out prior to registering *
*  a callback function for a Quickstep resource.  A pointer to  *
*  it is passed to the addResourceFilter() function           *
**************************************************************/
/*
typedef struct resourceInfo {
      int type;         // Type of resource to add filter to,
                        // RESOURCE_ANALOGIN, RESOURCE_ANALOGOUT, etc...
      int mode;         // Type of access to invoke filter on when accessed,
                        // RESOURCE_READ, RESOURCE_WRITE
      int start;        // Start number of resource (setting a range)
      int end;          // End number of resource, make same as start if
                        // only one (setting a range).
} RESOURCE_INFO;
*/
/**************************************************************
* FILTER_FUNCTION func                                        *
*                                                             *
* Type definition for Resource Callback function              *
*                                                             *
* PARAMETERS:                                                 *
* void *handle - handle returned by addUserResourceFilter when  *
*                function was registered                      *
* FILTERPARAMS *params - Access information block to detail   *
*                        what is being done                   *
* STDVAL value - Current value being read or written.         *
* RETVAL *status - pointer to status code that will be returned *
*                  to Quickstep.  Leaving it unchanged will   *
*                  default to SUCCESS.  Use only for defined  *
*                  errors.                                    *
*                                                             *
* RETURNS:                                                    *
* int - new value to return to Quickstep on a read or value to  *
*       write, if no change then return passed "value"        *
```

```
*****************************************************************/
void *addResourceFilter(RESOURCE_INFO *rsc, FILTER_FUNCTION func);

/*************************************************************************/
/*                                                                       */
/*  FUNCTION                                              RELEASE        */
/*                                                                       */
/*    removeResourceFilter                              PORTABLE C       */
/*                                                         1.0           */
/*  DESCRIPTION                                                          */
/*    This function is called to remove a resource filter from the       */
/*    Quickstep list.  A filter is removed by passing the handle that    */
/*    was returned when it was first added.                              */
/*                                                                       */
/*  INPUT                                                                */
/*    void *handle - Handle returned by addResourceFilter                */
/*                                                                       */
/*  OUTPUT                                                               */
/*    RETVAL -                                                           */
/*      SUCCESS = function called properly                               */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                 */
/*                                                                       */
/*  CALLS                                                                */
/*    Quickstep OS virtual table function pointer                        */
/*                                                                       */
/*  CALLED BY                                                            */
/*    As required by user code                                           */
/*                                                                       */
/*  RELEASE HISTORY                                                      */
/*                                                                       */
/*    DATE              NAME                      DESCRIPTION            */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
RETVAL removeResourceFilter(void *handle);
```

System Functions

RETVAL **enterStepAtomicity**(void);

RETVAL **exitStepAtomicity**(void);

```
/*************************************************************************/
/*                                                                       */
/*  FUNCTION                                              RELEASE        */
/*                                                                       */
/*    systemTics                                        PORTABLE C       */
/*                                                         1.0           */
/*  DESCRIPTION                                                          */
/*    This function is called to reset the watchdog system timer that    */
/*    will cause a system reset and fault if not invoked every 86 ms.    */
/*    Quickstep OS will automatically call this function as required     */
```

```
/*    but if a User Function runs as a step it must make the call if it   */
/*    maintains control too long, preventing Quickstep OS from calling    */
/*    the function.                                                       */
/*                                                                        */
/*  INPUT                                                                 */
/*    none                                                                */
/*                                                                        */
/*  OUTPUT                                                                */
/*    unsigned long - number of system tics, in milliseconds since powerup*/
/*                                                                        */
/*  CALLS                                                                 */
/*    Quickstep OS virtual table function pointer                         */
/*                                                                        */
/*  CALLED BY                                                             */
/*    As required by user code                                            */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
/*                                                                        */
/*                                                                        */
/**************************************************************************/
unsigned long systemTics(void);          // number of timer tics since
                                          powerup, 1ms/tic currently

/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                              RELEASE         */
/*                                                                        */
/*    fireWatchdog                                       PORTABLE C      */
/*                                                          1.0           */
/*  DESCRIPTION                                                           */
/*    This function is called to reset the watchdog system timer that     */
/*    will cause a system reset and fault if not invoked every 86 ms.     */
/*    Quickstep OS will automatically call this function as required      */
/*    but if a User Function runs as a step it must make the call if it   */
/*    maintains control too long, preventing Quickstep OS from calling    */
/*    the function.                                                       */
/*                                                                        */
/*  INPUT                                                                 */
/*    none                                                                */
/*                                                                        */
/*  OUTPUT                                                                */
/*    RETVAL -                                                            */
/*      SUCCESS = function called properly                                */
/*      ERROR_NOT_DEFINED = Quickstep OS table not found                  */
/*                                                                        */
/*  CALLS                                                                 */
/*    Quickstep OS virtual table function pointer                         */
/*                                                                        */
/*  CALLED BY                                                             */
/*    As required by user code                                            */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
```

```
/*                                                                      */
/*                                                                      */
/************************************************************************/
RETVAL fireWatchdog(void);
```

Threading

```
/************************************************************************/
/*                                                                      */
/*  FUNCTION                                                RELEASE     */
/*                                                                      */
/*    _tx_thread_create                                     PORTABLE C  */
/*                                                            1.0       */
/*  DESCRIPTION                                                         */
/*                                                                      */
/*    This function creates a thread and places it on the list of created */
/*    threads.                                                          */
/*                                                                      */
/*  INPUT                                                               */
/*                                                                      */
/*    thread_ptr                            Thread control block pointer  */
/*    name                                  Pointer to thread name string */
/*    entry_function                        Entry function of the thread  */
/*    entry_input                           32-bit input value to thread  */
/*    stack_start                           Pointer to start of stack     */
/*    stack_size                            Stack size in bytes           */
/*    priority                              Priority of thread (0-31)     */
/*    preempt_threshold                     Preemption threshold          */
/*    time_slice                            Thread time-slice value       */
/*    auto_start                            Automatic start selection     */
/*                                                                      */
/*  OUTPUT                                                              */
/*                                                                      */
/*    return status                         Thread create return status   */
/*                                                                      */
/*  CALLS                                                               */
/*    Quickstep OS virtual table function pointer                       */
/*                                                                      */
/*  CALLED BY                                                           */
/*    Application Code                                                  */
/*    _tx_timer_initialize                  Create system timer thread    */
/*                                                                      */
/*  RELEASE HISTORY                                                     */
/*                                                                      */
/*    DATE              NAME                        DESCRIPTION         */
/*                                                                      */
/************************************************************************/
unsigned int _tx_thread_create( TX_THREAD *thread_ptr,
      /* Task Name */
      char *name_ptr,
      /* Routine and Parameter to pass */
      void (*entry_function)(unsigned long), unsigned long entry_input,
                           /* Stack start and length */
      void *stack_start, unsigned long stack_size,
```

```
                                /* Priority and Threshold */
        unsigned int priority, unsigned int preempt_threshold,
        /* time slice */
        unsigned long time_slice, unsigned int auto_start);


/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                               RELEASE        */
/*                                                                        */
/*    _tx_thread_suspend                                   PORTABLE C
*/
/*                                                             1.0        */
/*  DESCRIPTION                                                           */
/*                                                                        */
/*    This function handles application suspend requests.  If the suspend */
/*    requires actual processing, this function calls the actual suspend  */
/*    thread routine.                                                     */
/*                                                                        */
/*  INPUT                                                                 */
/*                                                                        */
/*    thread_ptr                          Pointer to thread to suspend  */
/*                                                                        */
/*  OUTPUT                                                                */
/*                                                                        */
/*    status                              Return completion status      */
/*                                                                        */
/*  CALLS                                                                 */
/*    Quickstep OS virtual table function pointer                         */
/*                                                                        */
/*  CALLED BY                                                             */
/*    Application code                                                    */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
/*                                                                        */
/**************************************************************************/
unsigned int _tx_thread_suspend(TX_THREAD *thread_ptr);

/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                               RELEASE        */
/*                                                                        */
/*    _tx_thread_resume                                   PORTABLE C      */
/*                                                                        */
/*  DESCRIPTION                                                           */
/*                                                                        */
/*    This function processes application resume thread services.  Actual */
/*    thread resumption is performed in the core service.                 */
/*                                                                        */
/*  INPUT                                                                 */
/*                                                                        */
/*    thread_ptr                          Pointer to thread to resume   */
/*                                                                        */
/*  OUTPUT                                                                */
```

```
/*                                                                      */
/*    status                                  Service return status     */
/*                                                                      */
/*  CALLS                                                               */
/*    Quickstep OS virtual table function pointer                       */
/*                                                                      */
/*  CALLED BY                                                           */
/*    Application Code                                                  */
/*                                                                      */
/*  RELEASE HISTORY                                                     */
/*                                                                      */
/*    DATE              NAME                      DESCRIPTION           */
/*                                                                      */
/************************************************************************/
unsigned int _tx_thread_resume(TX_THREAD *thread_ptr);

/************************************************************************/
/*                                                                      */
/*  FUNCTION                                               RELEASE     */
/*                                                                      */
/*    _tx_thread_sleep                                    PORTABLE C    */
/*                                                           1.0        */
/*  DESCRIPTION                                                         */
/*                                                                      */
/*    This function handles application thread sleep requests.  If the  */
/*    sleep request was called from a non-thread, an error is returned. */
/*                                                                      */
/*  INPUT                                                               */
/*                                                                      */
/*    timer_ticks                             Number of timer ticks to sleep*/
/*                                                                      */
/*  OUTPUT                                                              */
/*                                                                      */
/*    status                                  Return completion status  */
/*                                                                      */
/*  CALLS                                                               */
/*    _tx_timer_activate                      Activate sleep timer      */
/*    _tx_thread_suspend                      Actual thread suspension  */
/*                                                                      */
/*  CALLED BY                                                           */
/*                                                                      */
/*    Application code                                                  */
/*                                                                      */
/*  RELEASE HISTORY                                                     */
/*                                                                      */
/*    DATE              NAME                      DESCRIPTION           */
/*                                                                      */
/************************************************************************/
unsigned int _tx_thread_sleep(unsigned long tics);

/************************************************************************/
/*                                                                      */
/*  FUNCTION                                               RELEASE     */
/*                                                                      */
/*    _tx_mutex_create                                    PORTABLE C    */
/*                                                           1.0        */
```

```
/*  DESCRIPTION                                                           */
/*                                                                        */
/*    This function creates a mutex with optional priority inheritance as */
/*    specified in this call.                                             */
/*                                                                        */
/*  INPUT                                                                 */
/*    mutex_ptr                         Pointer to mutex control block*/
/*    name_ptr                          Pointer to mutex name         */
/*    inherit                           Priority inheritance option   */
/*                                                                        */
/*  OUTPUT                                                                */
/*    TX_SUCCESS                      Successful completion status      */
/*                                                                        */
/*  CALLS                                                                 */
/*                                                                        */
/*    None                                                                */
/*                                                                        */
/*  CALLED BY                                                             */
/*    Application Code                                                    */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                        DESCRIPTION           */
/*                                                                        */
/**************************************************************************/
unsigned int _txe_mutex_create(TX_MUTEX *mutex_ptr, char *name, unsigned
int inherit);

/**************************************************************************/
/*                                                                        */
/*  FUNCTION                                                RELEASE       */
/*                                                                        */
/*    _tx_mutex_get                                       PORTABLE C      */
/*                                                           1.0          */
/*  DESCRIPTION                                                           */
/*                                                                        */
/*    This function gets the specified mutex.  If the calling thread      */
/*    already owns the mutex, an ownership count is simply increased.     */
/*                                                                        */
/*  INPUT                                                                 */
/*    mutex_ptr                       Pointer to mutex control block    */
/*    wait_option                     Suspension option                 */
/*                                                                        */
/*  OUTPUT                                                                */
/*    status                          Completion status                 */
/*                                                                        */
/*  CALLS                                                                 */
/*    _tx_timer_activate              Activate timer routine            */
/*    _tx_thread_suspend              Suspend thread service            */
/*    _tx_mutex_priority_change       Inherit thread priority           */
/*                                                                        */
/*  CALLED BY                                                             */
/*    Application Code                                                    */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
```

```
/*    DATE              NAME                        DESCRIPTION             */
/*                                                                            */
/******************************************************************************/
```

unsigned int _txe_mutex_get(TX_MUTEX *mutex_ptr, **unsigned long** wait_option);

```
/******************************************************************************/
/*                                                                            */
/*  FUNCTION                                                 RELEASE          */
/*                                                                            */
/*    _tx_mutex_put                                        PORTABLE C        */
/*                                                            1.0             */
/*  DESCRIPTION                                                               */
/*                                                                            */
/*    This function puts back an instance of the specified mutex.             */
/*                                                                            */
/*  INPUT                                                                     */
/*    mutex_ptr                         Pointer to mutex control block      */
/*                                                                            */
/*  OUTPUT                                                                    */
/*    TX_SUCCESS                        Success completion status           */
/*                                                                            */
/*  CALLS                                                                     */
/*    _tx_timer_deactivate              Deactivate timer routine            */
/*    _tx_thread_resume                 Resume thread service               */
/*    _tx_thread_system_return          Return to system routine            */
/*    _tx_mutex_priority_change         Restore previous thread priority    */
/*    _tx_mutex_prioritize              Prioritize the mutex suspension     */
/*                                                                            */
/*  CALLED BY                                                                 */
/*    Application Code                                                        */
/*                                                                            */
/*  RELEASE HISTORY                                                           */
/*                                                                            */
/*    DATE              NAME                        DESCRIPTION             */
/*                                                                            */
/******************************************************************************/
```

unsigned int _txe_mutex_put(TX_MUTEX *mutex_ptr);

```
/******************************************************************************/
/*                                                                            */
/*  FUNCTION                                                 RELEASE          */
/*                                                                            */
/*    _tx_mutex_delete                                     PORTABLE C        */
/*                                                            1.0             */
/*  DESCRIPTION                                                               */
/*                                                                            */
/*    This function deletes the specified mutex.  All threads                 */
/*    suspended on the mutex are resumed with the TX_DELETED status           */
/*    code.                                                                   */
/*                                                                            */
/*  INPUT                                                                     */
/*    mutex_ptr                         Pointer to mutex control block      */
/*                                                                            */
/*  OUTPUT                                                                    */
/*    TX_SUCCESS                        Successful completion status        */
```

```
/*                                                                      */
/*  CALLS                                                               */
/*    _tx_timer_deactivate              Deactivate timer routine        */
/*    _tx_thread_resume                 Resume thread service           */
/*                                                                      */
/*  CALLED BY                                                           */
/*    Application Code                                                  */
/*                                                                      */
/*  RELEASE HISTORY                                                     */
/*                                                                      */
/*    DATE              NAME                      DESCRIPTION         */
/*                                                                      */
/************************************************************************/
unsigned int _txe_mutex_delete(TX_MUTEX *mutex_ptr);
```

UDP Networking

```
/************************************************************************/
/*                                                                      */
/*  FUNCTION                                                RELEASE     */
/*                                                                      */
/*    commNetworkOpen                                      PORTABLE C   */
/*                                                            1.0       */
/*  DESCRIPTION                                                         */
/*     This function requests that a Network UDP or TCP port be openned.*/
/*     A RX thread will be spawned to automatically monitor the socket  */
/*     state.  This functions a USERCONNECTION control block that must  */
/*     not be modfied after a successful call and should be treated as  */
/*     a connection HANDLE on other Network calls.  A callback routine  */
/*     must be suppled as one of the USERCONNECTION parameters which will*/
/*     be invoked whenever a change of state occurs, including packet   */
/*     reception.                                                       */
/*                                                                      */
/*  INPUT                                                               */
/*    USERCONNECTION *user - Structure used to pass  connection/socket  */
/*                information, must be initialized prior to call.  Only */
/*                network type NETWORK_TYPE_UDP currently supported     */
/*  OUTPUT                                                              */
/*    int - 0 if success and -1 if failed                               */
/*      user->state set to USER_FAILED or USER_CONNECTING               */
/*      NOTE:  Once user->state becomes USER_CONNECTED this structure   */
/*             may be modifed and used for commNetworkSend operations   */
/*             It is cloned by the network thread.  The copy is what is */
/*             modified and supplied to the callback function.          */
/*  CALLS                                                               */
/*    Quickstep OS virtual table function pointer                       */
/*                                                                      */
/*  CALLED BY                                                           */
/*    As required by user code                                          */
/*                                                                      */
/*  RELEASE HISTORY                                                     */
/*                                                                      */
/*    DATE              NAME                      DESCRIPTION         */
/*                                                                      */
/*                                                                      */
/************************************************************************/
```

```
int commNetworkOpen( USERCONNECTION *user);

/*************************************************************************/
/*                                                                       */
/*  FUNCTION                                              RELEASE        */
/*                                                                       */
/*    commNetworkClose                                   PORTABLE C      */
/*                                                          1.0          */
/*  DESCRIPTION                                                          */
/*     This function requests that an existing network connection/socket */
/*     created with a call to commNetworkOpen, be closed.  The same      */
/*     USERCONNECTION structure passed to the commNetworkOpen call should */
/*     be passed to this function.                                       */
/*                                                                       */
/*  INPUT                                                                */
/*    USERCONNECTION *user - Structure used to pass  connection/socket   */
/*                information, must be initialized prior to call.  Only  */
/*                network type NETWORK_TYPE_UDP currently supported      */
/*                Should be same USERCONNECTION passed to commNetworkOpen */
/*                                                                       */
/*  OUTPUT                                                               */
/*    NONE                                                               */
/*    user->state modified to either USER_FAILED or USER_DISCONNECTED    */
/*                                                                       */
/*  CALLS                                                                */
/*    Quickstep OS virtual table function pointer                        */
/*                                                                       */
/*  CALLED BY                                                            */
/*    As required by user code                                           */
/*                                                                       */
/*  RELEASE HISTORY                                                      */
/*                                                                       */
/*    DATE              NAME                      DESCRIPTION            */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
void commNetworkClose( USERCONNECTION *user);

/*************************************************************************/
/*                                                                       */
/*  FUNCTION                                              RELEASE        */
/*                                                                       */
/*    commNetworkSend                                    PORTABLE C      */
/*                                                          1.0          */
/*  DESCRIPTION                                                          */
/*     This function requests that a packet be sent on a connection      */
/*     openned by commNetworkOpen.  The USERCONNECTION control block     */
/*     must contain the proper information.                              */
/*                                                                       */
/*  INPUT                                                                */
/*    USERCONNECTION *user - Structure returned by commNetworkOpen with  */
/*          following parameters set as desired:                         */
/*            packetBuf - pointer to buffer to send                      */
/*            length - number of bytes to send                           */
/*            destIp - If UDP, destination IP address                    */
/*            destPort - if UDP, destination port address                */
```

```
/*                                                                          */
/*  OUTPUT                                                                  */
/*    int - number of bytes queued                                          */
/*    user->state set to USER_FAILED or USER_DATAQUEUED                     */
/*      NOTE:  Limited queuing is available at the network stack level      */
/*             exceeding this will result in packet loss.  Do not send      */
/*             a large number of packets to the host without using a        */
/*             protocol that can verify packets have been transfered.       */
/*             This is typically only a problem if you attempt to sit in a*/
/*             loop continually transmitting data.                          */
/*  CALLS                                                                   */
/*    Quickstep OS virtual table function pointer                           */
/*                                                                          */
/*  CALLED BY                                                               */
/*    As required by user code                                              */
/*                                                                          */
/*  RELEASE HISTORY                                                         */
/*                                                                          */
/*    DATE              NAME                        DESCRIPTION             */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
int commNetworkSend( USERCONNECTION *user)
```

CHAPTER

4

UDPTERM/logEvent Utilities

UDPTerm is an unsupported tool which executes on your PC. It is extremely useful while trying to debug 'C' User programs. It communicates with the controller using UDP packets and listens for output, such as that from 'printf' statements. It can be found in the ..\tools\UDPTerm installation directory.

LogEvent is a function call that can be used to write 'unsigned long' values to a log buffer, for display from within UDPTerm. This is useful for low level debugging where slowing the processor down with a printf statement could cause a problem. In addition, 'logEvent is helpful when numerous writes are useful. It is also used internally by CTC to log any exceptions that may occur or network connections, reboots, and other helpful debugging information. A circular buffer is used. The function call consists of:

logEvent(EVENT_DEBUG,unsigned long); // where user supplies supply the unsigned long

When the controller is busy, 'printf' output may not always appear so if there is any question you will want to use the logEvent(EVENT_DEBUG,####); routine.

Invoking UDPTerm

To invoke UDPTerm, on your PC the IP address of the terminal and the port to send and listen on is required. Only port 1202 is supported by the controller. The following example shows how to connect to a controller with the IP address of 12.40.53.158:

UDPTerm 12.40.53.158 1202 1202

After contact is made press the Enter key a few times and the 'Command:' prompt will appear. There are numerous commands, some of the useful ones are:

l - list log, logEvent values can be viewed with this command. For example in the 'Network Startup' thread the #00009 log was done with a logEvent(EVENT_DEBUG,3015); function call.

```
Command : l

Processing : l ()
Events=82 (t=3750)
#00001 (Not Thread) at 00000000 00000 (0x18010) Clear Log, 0x00000001(1)
#00002 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x06083908) Disable
#00003 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x060839d8) No Expan
#00004 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x06083a24) No Expan
#00005 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x060839d8) No Expan
#00006 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x06083a24) No Expan
#00007 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x060839d8) No Expan
#00008 (Not Thread) at 00000000 00000 (0xdf01) Diagnostic, (0x06083a24) No Expan
#00009 (Network Startup) at 00000001 00000 (0x8f00) Diagnostic, 0x00000bc7(3015)
#00010 (Network Startup) at 00000001 00000 (0x8f00) Diagnostic, 0x00000bc8(3016)
#00011 (Network Startup) at 00000002 00000 (0xdf01) Diagnostic, (0x06089724) No
#00012 (Network Startup) at 00000002 00000 (0xdf01) Diagnostic, (0x06089714) Con
#00013 (Network Startup) at 00000002 00000 (0xdf01) Diagnostic, (0x06089708) Phy
#00014 (Network Startup) at 00000002 00000 (0x8f00) Diagnostic, 0x00000bc9(3017)
#00015 (QS Main) at 00000003 00000 (0x8f00) Diagnostic, 0x00001f55(8021)
#00016 (QS Main) at 00000004 00000 (0x8f00) Diagnostic, 0x00001f55(8021)
#00017 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00018 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00019 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00020 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00021 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00022 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x060839d0) Slot ID, 0x
#00023 (QS Main) at 00000004 00000 (0xdf01) Diagnostic, (0x06083d1c) No Expansio
#00024 (Network Startup) at 00000052 00000 (0x8f00) Diagnostic, 0x00000bca(3018)
#00025 (Network Startup) at 00000052 00000 (0x8f00) Diagnostic, 0x00000bcb(3019)
#00026 (Network Startup) at 00000052 00000 (0x8f00) Diagnostic, 0x00000bcc(3020)
#00027 (Network Startup) at 00000052 00000 (0x8f00) Diagnostic, 0x00000bcd(3021)
```

L X - clear log

```
Command : L X

Processing : L (X)
Log Cleared
Command :
```

o ? - thread help

```
Command : o ?

Processing : o (?)
A = all tasks
C = counters
P = pools
M = mutexes
Q = queues
S = semaphores
Tn = task n
U = memory Used
Rn= resume thread
Hn = halt thread
Command :
```

o - list threads running summary

```
Command : o

Processing : o ()
Unrecognized command , displaying status

Status : 0
Ready Bits : 00001020

Ticks=88572186
Tasks :
062b571c System Timer Thread Suspended
062c0ad4 QS Main Ready to run
062c022c Thread Monitor Suspended
062a986c Network Startup Waiting on Delay
062bd474 CTC_BF_10063261 Waiting on Event Flag
062b15ac Command Processor Ready to run
062c3548 GDB thread Suspended
062bc2ac Analog Input Scanner Waiting on Delay
062bfd24 CTC_BF_10063261 Waiting on TCP/IP
06216f9c UDP Peer to Peer RX Waiting on TCP/IP
0621807c TCP Modbus Slave Suspended
0621915c UDP Peer to Peer TX Waiting on Queue
0621a94c UDP Binary Protocol RX Waiting on TCP/IP
0621ba2c TCP Binary Server Suspended
0621cb0c SNTP Client Waiting on Delay
0621dbec CTServer Broadcast RX Waiting on TCP/IP
0621f36c FTP Server Suspended
0622044c TELNET Server Suspended
06221c9c Web Thread Suspended
0622511c UNUSED Suspended

Command :
```

o m - list mutexs and what is pending

```
Command : o m

Processing : o (m)
Total Mutexes 33
Mutex 'Malloc Mutex' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Local printf Mutex' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Resource Filter Mutex' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Quickstep Safe' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Step Atomicity' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'CTC_BF_10063261' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'SPI Select' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Parse Binary' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Comm0 General' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Volatile Register Put' Id : 4d555445, Pending tasks 0
  Owner = None
Mutex 'Volatile Register Get' Id : 4d555445, Pending tasks 0
```

o t - list threads and detailed information

```
Command : o t

Processing : o (t)
All Tasks :
Task 'System Timer Thread' at 62b571c
Priority 0, Threshold 0
Status (3) Suspended
Count 34387 Delay 0
Stack 62ca4c0:62ca8bf, 62ca854
Short Stack Frame 0
R8-R14, MACH,L, PR, SR
  062b15f8 00000001 062ca888 062b15ac 062b57b4 062b5668 062ca884 00000000
  00000000 0600f23a 00000001

Task 'QS Main' at 62c0ad4
Priority 12, Threshold 12
Status (0) Ready to run
Count 38324 Delay 0
Stack 62c0b6c:62c1b5b, 62c1aa8
Long Stack Frame 1
R0-R14, MACH,L, PR, PC, SR
  00000001 062aa948 06023170 0602b290 0604bfa0 062aa91c 1001e00c 00000000
  0602318e 1001e010 1001e00c 00000000 1001e00c 00000002 00000028 062c1b04
  00000000 0604c322 062c1b08 062c1b20
```

Net ? – network help

```
Command : net ?

Processing : net (?)
I (ad) = IP Address
M (ad) = Subnet Mask
G (ad) = Gateway
N (ad) = CTC Node
H (ad) = NIC Address
C Counters
S Sockets Open
D Display all
Command :
```

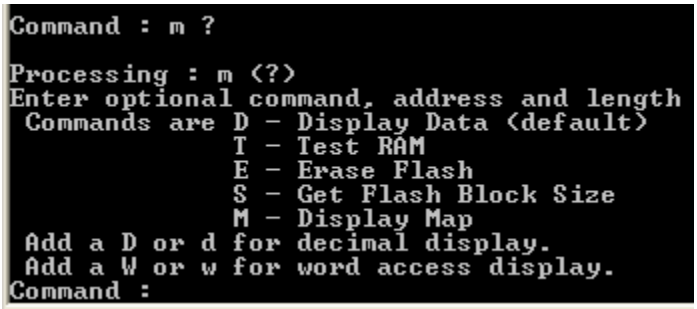
net - network state and information, IP, MAC...

```
Processing : net ()
Unrecognized command , displaying all
IP Address (c283561), 012.040.053.097
Subnet Mask (ffffff00), 255.255.255.000
Gateway (c2835cc), 012.040.053.204
CTC Node 0, Working 0
NIC Address 00c0-cb99-8d9d
Command :
```

dhcp - dhcp information

```
Command : dhcp

Processing : dhcp ()
Unrecognized command , Display status

  DHCP Status
Xid 93c87970, State : Bound
My IP      = 012.040.053.097
Server IP = 012.040.053.008
Packet Pool : CTC_BF_10063261, Socket : CTC_BF_10063261,
Lease 86400 24:00:00, Renew 43200 12:00:00, Rebind 75600 21:00:00
Timeout 43200 12:00:00, left -1

Current IP = 012.040.053.097 SubNet = 255.255.255.000 Gateway = 012.040.053.204

Ethernet Flags 0
Command :
```

m ?



m 0xaddress - dump memory starting at 0x######## in hex

All of these commands are the same as used when using telnet and run the 'enable debug' command. Also only one instance may run at a time on the PC.

Blank

CHAPTER

5

UserApp.c Sample Program

This section includes sample code which is distributed within the ‘C’ Development kit for the controller. This program introduces the concepts of register filters, serial and network communications.

```
/*****************************************************************************/
/* MODULE: userApp.c                                                         */
/*                                                                           */
/* MODULE DESCRIPTION:                                                       */
/*                                                                           */
/* This module is provided as an example for accessing the Quickstep OS      */
/* from a C programming environment.  The supplied main() example is for     */
/* reference only and must exist somewhere in the users code and it must     */
/* return control in a timely fashion.                                       */
/*                                                                           */
/* REVISION HISTORY:                                                         */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

// Always include CoreFunc.h in any module that will access the Quickstep OS
#include "CoreFunc.h"

#define ESC "\x1B"

// Function prototype
int sampleFilter(void *handle, FILTERPARAMS *params, STDVAL value, RETVAL *status);

/*****************************************************************************/
/*                                                                           */
/*  FUNCTION                                                 RELEASE         */
/*                                                                           */
/*    main                                                 PORTABLE C       */
/*                                                            1.0            */
/*  DESCRIPTION                                                              */
/*                                                                           */
/*    This function is the main input function called after a User           */
/*    C file is loaded into memory for execution.  Any initialization        */
/*    required should be done and control returned.                          */
/*                                                                           */
/*    MAKE SURE TO RETURN CONTROL AND DON"T TAKE LONG!!!  IF CALLED BY       */
/*    A QUICKSTEP RUNNING A SCRIPT, YOU MAY HAVE TO INVOKE A WATCHDOG        */
```

```
/*    RESET FUNCTION IF SPEND MORE THAN 40 MS HERE  (fireWatchdog())      */
/*                                                                        */
/*  INPUT                                                                 */
/*                                                                        */
/*    none                                                                */
/*                                                                        */
/*  OUTPUT                                                                */
/*                                                                        */
/*    0 = Initialization successful, allow User functions to execute      */
/*    non-zero = Init failed, do not run User functions                   */
/*                                                                        */
/*  CALLS                                                                 */
/*    - __main() call must be madie prior to any other code               */
/*    - user define initialization routines, as required                  */
/*                                                                        */
/*  CALLED BY                                                             */
/*                                                                        */
/*    C User Function file loader (Quickstep OS)                          */
/*                                                                        */
/*  RELEASE HISTORY                                                       */
/*                                                                        */
/*    DATE              NAME                      DESCRIPTION             */
/*                                                                        */
/*                                                                        */
/**************************************************************************/
int __main();

int main (void)
{
        RESOURCE_INFO resource;
        void *handle;

        if (__main() < 0)
        {
                return -1;              // Can't find link to main function table so must
                                        // return control and report error
        }

        // Log Event Code to signal we made it here successfully.  This is not
        // required but placed here as test code.  Note all user Events should
        // start at 50000, below that is reserved for Quickstep OS
        logEvent(EVENT_DEBUG, 50000);

        // Send a message to the UDP Debugger screen as though it was our STDIO output
        printf("Hello World.\r\n");

        // Lets install a filter function as a sample
        resource.type = RESOURCE_REGISTER;
        resource.start = 2;     // lets filter register 2
        resource.end = 6;       // no range, only 2 for now
        resource.mode = RESOURCE_READ;        // read operation only, if write too would |
RESOURCE_WRITE
        // Now add the filter...
        handle = addResourceFilter(&resource,sampleFilter);

        // if error occurs return non-zero and User functions will not run...
        return(0);
}

// User Filter Example, register 2 to 6 will be divided by 2 when written
int sampleFilter(void *handle, FILTERPARAMS *params, STDVAL value, RETVAL *status)
{
        // This sample filter simply processes a read or write operation on a register
        switch(params->mode)
        {
        case RESOURCE_READ:
                // Someone is attempting to read the register, actual value in in "value"
                // Let's divide it by 2 just for test purposes
                if (value)      // don't divide by 0...
                        value = value/2;
                break;
```

```
        case RESOURCE_WRITE:
                // Let's not do anything on a write operation, we could have added filter
                // so only reads called this funtion also but this is for future reference
                // on how to process a write
                break;
        }
        // Return new or same value to use
        return value;
}

// User taskLoop Function Example invoked on main Quickstep loop after all steps execute
// NOTE: THIS ONLY RUNS WHEN CONTROLLER IS NOT FAULTED!!!
//       EVEN IF QUICKSET IS NOT USED NEED A SINGLE INSTRUCTION QUICKSTEP PROGRAM IN THE
CONTROLLER!
//  userServiceLoopHook is called when faulted and also after every task step
//  you may use the SYSMODE state to determine when taskLoop is not running, possibly only
running
//  certain tasks when in FAULT mode, like communications.  Note that if faulted power is
//  is turned off on the outputs (VBIAS).
int taskLoop(SYSMODE state)
{
        void testSerial(int);
        void testNetwork();
        // Note spending more than 40 ms in this loop will cause a watchdog fault!  Not
        // including time may be pre-empted by interrupts and communication threads
        // fireWatchdog();  will need to be called to reset timer if do.
        // (Watchdog reset upon entry and exit of this function, automatically)
        testSerial(COM1);
        testNetwork();
        return 0;       // always return 0, not used but may be some day
}

// Serial port test function
// commPort = COM1 or COM2 for hardware serial ports
// Register 1 = transmit status/results
// Register 2 = loop counter
void testSerial(int commPort)
{
static int initialized = 0;
static int i = 1;
static int txcnt = 1;
static int errcnt = 0;
RETVAL r;
UINT8 szMsg[3];
UINT16 wSize;
char buf[70];

        if (!initialized)
        {
                initialized = 1;
                // first must disable parsing so get raw data on receiver
                szMsg[0] = COMMCMD_RQST_PARSING;
                szMsg[1] = 0;  /*dummy for port value*/
                szMsg[2] = 0;
                wSize = 3;
                commGenericCmd(commPort, szMsg, &wSize);
        }

        // Invoked here on each loop of quickstep execution
        // Check to see if transmitter is ready
        szMsg[0] = COMMCMD_RQST_QUERY;
        szMsg[1] = 0;  /*dummy for port value*/
        wSize = 2;
        commGenericCmd(commPort, szMsg, &wSize);

        if (szMsg[0] == COMMCMD_RSPN_OK)
        {
                // Serial port ready
                sprintf(buf,"Control Technology Serial Port Test #%d, errors -
%d.\r\n",txcnt,errcnt);
                r = commSendMsg(commPort, buf,strlen(buf));
```

```
                if (r != SUCCESS)
                {
                        // error occured, r = 35 ERROR_NO_COMM_PORT
                        errcnt++;
                        regWrite(1, r);
                }
                else
                {
                        txcnt++;
                        regWrite(1, 0);
                }
        }
        else if (szMsg[0] == COMMCMD_RSPN_BUSY)
        {
                // port is busy
                regWrite(1, 999);
        }
        else
        {
                // unknown response
                regWrite(1,888);
        }
        // set loop counter, using register 18
        regWrite(2, i++);
}

/****************** BELOW FOR NETWORK OPERATION *********************/
USERCONNECTION user;   // Structure to define network connections
                                                // must not be stack variable and must be passed
                                                // to Open, Send, and Close routines

// Maximum number of packets that can be queued
#define MAX_USER_PACKETS_QUEUED 5
#define MAX_USER_MESSAGE_SIZE 400 // set this to largest message size
// Packet structure for queue
typedef struct
{
        unsigned char packet[MAX_USER_MESSAGE_SIZE+1];
        int length;
        unsigned long hostIp;
        int hostPort;
} NETWORKPACKET;

// When counts -1 there are no packets to process, when same full
int incount;   // storage count
int outcount;  // retrieval count
NETWORKPACKET packets[MAX_USER_PACKETS_QUEUED];

TX_MUTEX packetMutex;  // Need to mutex joint resources with main Quickstep thread

// Network Function test program to create a thread to monitor UDP port 7000
// also sends a sample message upon connection
void testNetwork()
{
static int state = 0;
void networkStateChange(void *ptr);

        if (state == 0)
        {
                // first time called therefore open a UDP port, 7000
                // clear everything out
                memset((void *)&user,0,sizeof(USERCONNECTION));
                user.srcPort = 7000;
                user.type = NETWORK_TYPE_UDP;
                incount = outcount = -1;      // set to empty
                // define the function to be invoked by network thread when change of state
                // Note that mutexes must be used within this function if common resources
                // accessed
                user.StateCallback = networkStateChange;
                printf("attempt open.\r\n");
                // Create a mutex since this thread and network callback are different threads
```

```
            _txe_mutex_create(&packetMutex,"USER C UDP",TX_INHERIT);
        if (commNetworkOpen((USERCONNECTION *)&user))
        {
                // error occurred, cleanup
                printf("Network Open failed\r\n");
                _txe_mutex_delete(&packetMutex);
                return;
        }
        printf("open occurred.\r\n");
        // Network thread spawned so now await callback with change of state
        // saying connected.  Can also watch our Control Block.
        state++;
}
else if (state == 1)
{
        // await Network thread to be operational
        if (user.state == USER_CONNECTED)
        {
                printf("got connection.\r\n");
                // we are now ready to run
                state++;
        }
        else if (user.state == USER_DISCONNECTED)
        {
                // connection failed, abort, already closed
                state--;
        }
        else if (user.state == USER_CONNECTING)
        {
                // thread still starting up
        }
        else
        {
                // undefined?????????
        }
}
else if (state == 2)
{
        // lets idle since running, wait for Network data to process
        if (outcount == -1)
        {
                // nothing to do
                return;
        }
        // make sure it is null terminated
        packets[outcount].packet[packets[outcount].length] = 0x00;
        // get next buffer to use, do something
        printf("%s\r\n",packets[outcount].packet);   // this sent to UDPterm last heard
                                                     // from on port 1202

        // Test sending a response to the packet
        user.packetBuf = "UDP Packet reception Successful";
        user.length = strlen(user.packetBuf);
        // set response destination to originator of packet
        user.destIp = packets[outcount].hostIp;
        user.destPort = packets[outcount].hostPort;
        commNetworkSend((USERCONNECTION *)&user);

        // must set mutex when bump counter but not needed to process data
        _txe_mutex_get((TX_MUTEX *)&packetMutex, TX_WAIT_FOREVER);
        outcount++;
        if (outcount == MAX_USER_PACKETS_QUEUED)
        {
                outcount = 0;
        }
        if (outcount == incount)
        {
                // we are empty now
                outcount = incount = -1;
        }
        // unlock resource
```

```
                _txe_mutex_put((TX_MUTEX *)&packetMutex);

        }
}

// Callback function to process RX thread state change
void networkStateChange(void *ptr)
{

// It is left to the user to modify the below code for their application

        USERCONNECTION *user = (USERCONNECTION *)ptr;

        switch(user->state)
        {
                case USER_CONNECTED:
                        // OK to send now
                        printf("Got USER_CONNECTED\r\n");
                        break;

                case USER_DISCONNECTED:
                        // this connection is has been broken
                        printf("Got USER_DISCONNECTED\r\n");
                        break;

                case USER_DATA_AVAILABLE:
                        // offload data packet, do not access serial ports except on may
                        // Quickstep loop

                        // the routines is not thread safe
                        printf("Got USER_DATA_AVAILABLE, length %d\r\n",user->length);
                        // move data to buffers
                        _txe_mutex_get((TX_MUTEX *)&packetMutex, TX_WAIT_FOREVER);
                        if (incount == -1)
                        {
                                // this is the first packet
                                outcount = 0;
                                incount = 0;
                        }
                        else if (incount == outcount)
                        {
                                _txe_mutex_put((TX_MUTEX *)&packetMutex);
                                return;         // ignore packet, buffers full
                        }
                        // Move the packet data into our local queues for processing
                        memcpy(packets[incount].packet,user->packetBuf,user->length);
                        packets[incount].length = user->length;
                        packets[incount].hostIp = user->destIp;
                        packets[incount].hostPort = user->destPort;
                        incount++;
                        if (incount == MAX_USER_PACKETS_QUEUED)
                        {
                                // wrap it
                                incount = 0;
                        }
                        _txe_mutex_put((TX_MUTEX *)&packetMutex);
                        break;

                default:
                        printf("Got undefined state\r\n");
                        return;
        }
}

// Function called when program is being unloaded, possibly new being loaded
void userCleanUp()
{
// must return any allocated memory to system or close network conn.if open so threads stop
        if (user.state == USER_CONNECTED)
        {
                commNetworkClose((USERCONNECTION *)&user);
```

```
        }
        _txe_mutex_delete((TX_MUTEX *)&packetMutex);
}
```