

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Keysight / Agilent E6237A* at our website: ***Click HERE***

HP E6237A
Compiled SCPI for LynxOS

User's Guide

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard Company (HP) shall not be liable for any errors contained in this document. *HP makes no warranties of any kind with regard to this document, whether express or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.* HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Warranty Information

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

U.S. Government Restricted Rights

The Software and Documentation have been developed entirely at private expense. They are delivered and licensed as “commercial computer software” as defined in DFARS 252.227-7013 (Oct 1988), DFARS 252.211-7015 (May 1991) or DFARS 252.227-7014 (Jun 1995), as a “commercial item” as defined in FAR 2.101(a), or as “Restricted computer software” as defined in FAR 52.227-19 (Jun 1987) (or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and Documentation by the applicable FAR or DFARS clause or the HP standard software agreement for the product involved.

Copyright © 1997 Hewlett-Packard Company. All rights reserved.

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intel 486™ is a U.S. trademark of Intel Corporation.
Pentium® is a U.S. registered trademark of Intel Corporation.
UNIX® is a registered trademark in the U.S. and other countries, licensed exclusively through X/Open Company Limited.

Printing History

Edition 1 - August 1997

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contents

1. Getting Started with Compiled SCPI

Conventions Used....2
Verifying your Compiled SCPI System....3
Preparation Check List....3
Reviewing System Components....3
Verify System Setup....4
Running your First Compiled SCPI Program....6
Using the Getting Started Tutorial....6
Review the C-SCPI Process....8
Reviewing getstrt1.cs Program....9
Accessing C-SCPI Online Documentation....10
Learning About Compiled SCPI....11
Advantages of Using Compiled SCPI....12
Compiled SCPI System....13
Maximum System Throughput with Compiled SCPI....14

2. Using Compiled SCPI

Overview of C-SCPI....16
Creating Source Code....18
Defining the C-SCPI Commands....19
Running the C-SCPI Preprocessor....20
Compiling/Linking your Code....22
Executing Your Program....25
Using Makefiles....25
Preprocessor Options....27
The Compiled SCPI Preprocessor Command....27
Storing Block Data in a Separate File (The -f Option)....27
Using SCPI Only Files (The -i Option)....29
Interactive Functions....32
Using Interactive Functions....32
Programming with Interactive Functions....34
Overlapped Mode....36
Determining if you Should use the Overlapped Mode....36

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2. Using Compiled SCPI (*continued*)

Throughput and the Overlapped Mode 37
Overlapped Command..... 40
Using the Overlapped Mode 40
Controlling Overlapped Execution..... 43
Programming for Efficiency..... 45

3. Programming with Compiled SCPI

Looking at an Example System Configuration 48
Providing an Error Routine 50
Programming with a Scanning Multimeter 51
Programming with an External File 53
Programming with a C-SCPI Parameter List 56
Storing Block Data in a Separate File (The -f Option) 59
Using C-SCPI in the Interactive Mode 62
Triggering with the HP Embedded Computer..... 64

4. Troubleshooting Compiled SCPI

Resolving Compiled SCPI Preprocessor Errors..... 66
Resolving Compile and Link Errors..... 69
Resolving Compiled SCPI Run-Time Errors 74
Using GNU Debugger 79
Trapping Errors with cscpi_error 81

5. Compiled SCPI Command Reference

INST_CLEAR 88
INST_CLOSE..... 90
INST_DECL..... 92
INST_EXTERN 94
INST_ONSRQ..... 96
INST_OPEN..... 98
INST_PARAM 100
INST_QUERY 102
INST_READSTB 108
INST_SEND..... 110
INST_STARTUP 116

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contents

5. Compiled SCPI Command Reference (*continued*)

INST_TRIGGER 117
cscpi_error 119
cscpi_exe 120
cscpi_exe_fildes 122
cscpi_exe_stream 124
cscpi_get_overlap 126
cscpi_overlap 127
Compiled SCPI Quick Reference 128

A. Online Documentation

How To Use Manual Pages 132

B. Compiled SCPI Software Installation

C. Compiled SCPI File Structure

C-SCPI Directories 140
Other Directories 140
Structure for C-SCPI on LynxOS 141

D. Threads and Compiled SCPI

Writing your Compiled SCPI Programs 144
Compiling your Compiled SCPI Programs 144

E. Error Messages

Compiled SCPI Preprocessor Errors 146
Compile and Link Errors 147
Run-Time Errors 148

F. Other Documentation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1

Getting Started with Compiled SCPI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Getting Started with Compiled SCPI

Welcome to the Compiled SCPI (Standard Commands for Programmable Instruments) User's Guide. Throughout this guide Compiled SCPI is called C-SCPI. This guide provides detailed information about how to use Compiled SCPI. Example Compiled SCPI programs and troubleshooting information is provided as well. For information about specific VXI instruments, see the instrument's user's manual or the online documentation provided.

Conventions Used

This guide uses the following conventions:

Notation	Description
non-italics	Non-italicized words within the **Syntax** description are literals. That is, enter them exactly as shown. This includes non-italicized braces and brackets. Non-italicized words and punctuation appear in `COMPUTER FONT`.
italics	Italicized words within the **Syntax** description represent argument names, program names, or strings that you must replace with an appropriate value.
[ ]	Brackets within the **Syntax** description determine optional elements.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Verifying your Compiled SCPI System

To ensure that you are ready to start programming your system using the Compiled SCPI software, you should review the system check list for hardware and software requirements, learning products, verification of your system setup, and how to run a simple program. The following lists the steps in preparing for a successful experience.

Preparation Check List

1. Review System Components
   -- Hardware Requirements
   -- Software Requirements
   -- Learning Products
2. Verify your System Setup
3. Run your First C-SCPI Program (see “Running your First Compiled SCPI Program” on page 6)
   -- Define C-SCPI Process
   -- Run the C-SCPI Preprocessor
   -- Create Executable Code
   -- Execute your Program
   -- Access C-SCPI Online Help

Reviewing System Components

This section lists the system components you should have upon receipt of your system. The requirements for this system include hardware, software, and learning products or documentation.

Hardware Requirements

The minimum hardware requirements for your system are as follows:

- HP Pentium® embedded controller with:
  -- minimum 1GB disk drive
  -- 16MB DRAM

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Software Requirements

The software requirements for your system are as follows:

- LynxOS Real-Time operating system
- HP SICL software for LynxOS
- HP Compiled SCPI
  - -- Preprocessor
  - -- all instrument drivers

Learning Products

The learning products you should have with your system include the following:

- LynxOS complete manual set
- HP SICL complete manual set
- HP Compiled SCPI
  - -- HP Compiled SCPI User's Manual (this document)
  - -- HP Compiled SCPI Online Documentation
  - -- HP Compiled SCPI Example programs (`/usr/hp75000/demos/cscpi`)

Verify System Setup

This section provides commands to verify that the required software is installed. See Appendix B for C-SCPI installation instructions.

LynxOS Real-Time Operating System

At the command prompt, type the following command:

```
uname -r
```

This will display the version number of the operating system. Release 2.5.0 or later of LynxOS Real-Time is required to run the C-SCPI software. You should see something similar to the following:

```
2.5.0
```

SICL for LynxOS

At the command prompt, type the following command:

```
devices
```

In the list, you should see the following device:

```
sicl_driver
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C-SCPI Software

At the command prompt, type the following:

```
ident /lib/libcscip.a
```

When you execute this command, information for the C-SCPI preprocessor and each instrument driver installed on your system is listed:

- name of C-SCPI software or Driver
- version number

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Running your First Compiled SCPI Program

This section provides a guide to using HP Compiled SCPI (C-SCPI) commands within your C program and to running the C-SCPI preprocessor.

Note

This chapter focuses on using C-SCPI Commands and its preprocessor. It does **not** focus on teaching you how to program using the GNU CC Compiler provided by Lynx Real-Time Systems Inc.

A tutorial is provided to step you through the C-SCPI process of creating executable code. It uses a simple C program with C-SCPI commands to get you up and running quickly. This program does not attempt to do anything very ambitious. Task-oriented programs are provided in Chapter 3.

Using the Getting Started Tutorial

This section steps you through the process for creating C-SCPI executable code.

1. Copy the getstrt1.cs program to your directory

   -- The program, `getstrt1.cs`, is located in the directory `/usr/hp75000/demos/cscpi`. Once you have copied the sample program into your own directory, you can treat it as you would any C program. Figure 2-1 lists `getstrt1.cs` in its entirety for your convenience, and is displayed later in this chapter. This example uses a Hewlett-Packard multimeter (HP E1411B instrument) at logical address 24. (Note: if your multimeter is NOT at logical address 24, you must change the address in the C-SCPI example program. See “Reviewing getstrt1.cs Program” on page 9.)

2. Run the C-SCPI Preprocessor

   -- Run the C-SCPI preprocessor to translate your C-SCPI commands into C function calls by typing the following at the command prompt:

   ```
   cscpipp getstrt1.cs > getstrt1.c
   ```

3. Compile your program

   -- Compile your program by typing the following at the command prompt:

   ```
   gcc -c [-g] -mthreads getstrt1.c
   ```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comments

- The **`-c option`** creates a file called `getstrt1.o`
- The **`-g option`** indicates to the compiler that the output should contain debug information (this is optional)
- The **`-mthreads option`** indicates threaded functions are being used
- The **`-I option`** causes the listed path to be searched for more include files

4. Link your program

-- Link your program by typing the following at the command prompt:

```
gcc [-g] -mthreads -o getstrt1 getstrt1.o
   -lcscpi -lsicl
```

Comments

- The **`-g option`** indicates to the linker that the debugger is being used
- The **`-mthreads option`** indicates threaded functions
- The **`-o option`** creates an executable file called getstrt1
- The **`-l options`** link in the C-SCPI and SICL libraries

5. Execute your program

-- Execute your program by typing the following at the command prompt:

```
getstrt1
```

The result of executing this program should be **SIMILAR** to the following:

```
The id of this module is HEWLETT-PACKARD,
 E1411B, 0,E.05.02
```

Note

To Compile and Link in one step, refer to “Compiling/Linking your Code” in Chapter 2. Also, you can use a makefile to run the preprocessor, compile and link your programs. Refer to “Using Makefiles” in Chapter 2 for information on using a makefile.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Review the C-SCPI Process

The following flowchart defines the C-SCPI process to create executable code.

1. Copy the getstrt1.cs program or Write a C program with C-SCPI commands

```
/*getstrt1.cs
#include <cscpi.h>
INST_DECL (vm,"E1411B",REGISTER);
main ()
{
  INST_STARTUP()
  INST_OPEN(vm,"vxi,24");
  INST_SEND(vm,"*RST");

}
```

2. Run the C-SCPI Preprocessor

```
cscpipp getstrt1.cs > getstrt1.c
```

3. Run your C compiler

```
gcc -c [-g] -mthreads getstrt1.c
```

4. Link your code with C-SCPI and SICL libraries

```
gcc [-g] -mthreads -o getstrt1 getstrt1.o
   -lcscpi -lsicl
```

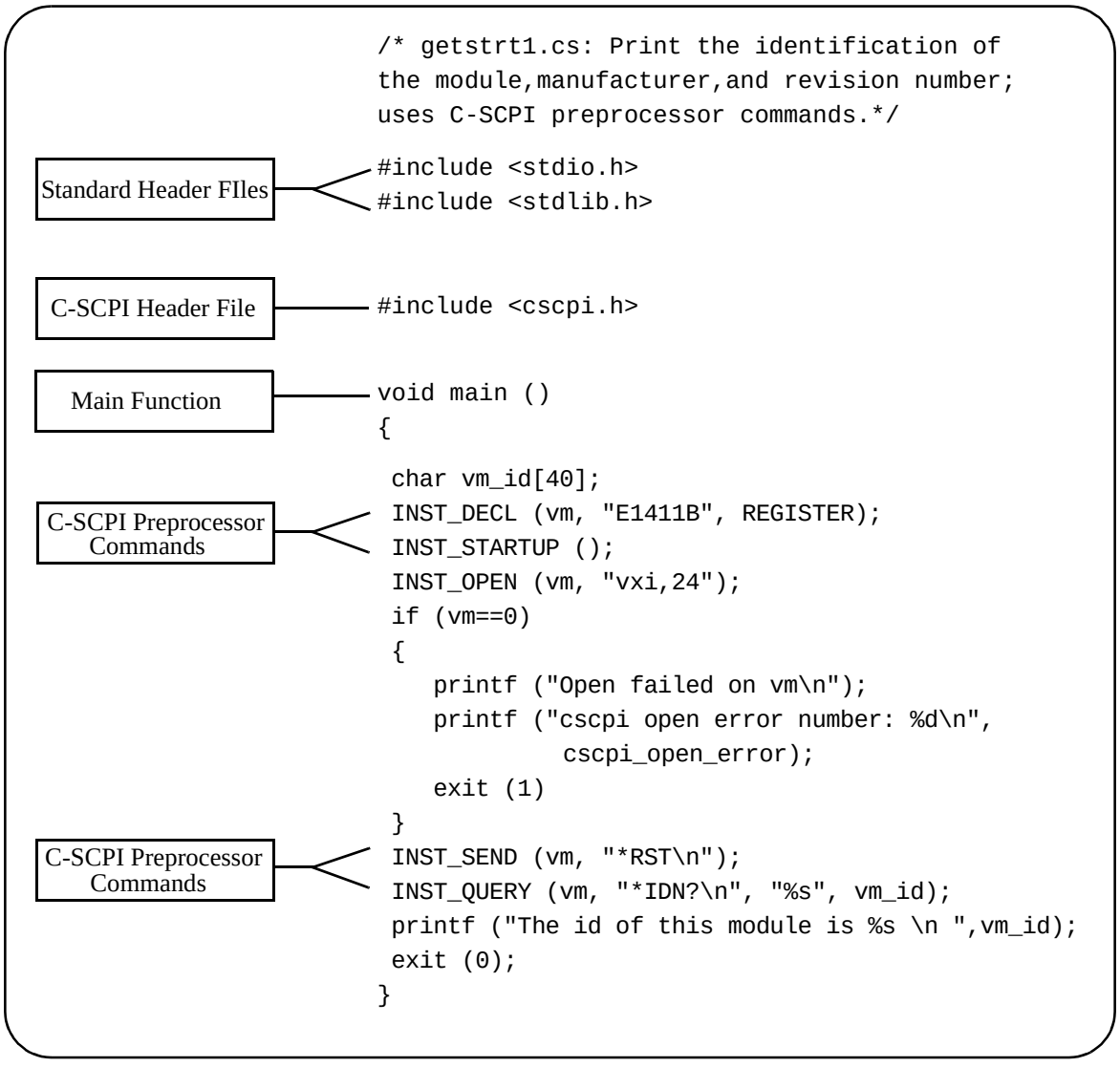
5. Run the executable code

```
getstrt1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Reviewing getstrt1.cs Program

Figure 1-1 lists the `getstrt1.cs` program in its entirety for your convenience.




```
/* getstrt1.cs: Print the identification of
the module,manufacturer,and revision number;
uses C-SCPI preprocessor commands.*/

#include <stdio.h>
#include <stdlib.h>

#include <cscpi.h>

void main ()
{
 char vm_id[40];
 INST_DECL (vm, "E1411B", REGISTER);
 INST_STARTUP ();
 INST_OPEN (vm, "vxi,24");
 if (vm==0)
 {
    printf ("Open failed on vm\n");
    printf ("cscpi open error number: %d\n",
             cscpi_open_error);
    exit (1)
 }
 INST_SEND (vm, "*RST\n");
 INST_QUERY (vm, "*IDN?\n", "%s", vm_id);
 printf ("The id of this module is %s \n ",vm_id);
 exit (0);
}
```

Figure 1-1. getstrt1.cs C Program

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you do not have an HP E1411B instrument installed in your C-size mainframe, you will need to change the following lines in the example program:

-- INST_DECL(vm,"**E1411B**",**REGISTER**);

- change the instrument driver to your instrument
  E1411B ⟶ E1330, E1446, E1414, etc.
- check the instrument mode
  REGISTER ⟶ REGISTER OR MESSAGE

-- INST_OPEN(vm,"vxi,**24**");

- change the logical address to your instrument's logical address

Accessing C-SCPI Online Documentation

To access online documentation via manual pages, type the following at the command prompt (remember, `man` is case sensitive):

```
man name
```

-- where *name* is replaced by the following:

C-SCPI Command, (e.g., INST_DECL)
Instrument driver, (e.g., E1411B)

For a complete list of instrument drivers available, type the following at the command prompt:

```
man cscpi_drivers
```

You can also use the `cscpipp -?` command to find out what drivers are installed on your system.

Additional Information

For additional information on accessing C-SCPI online documentation, refer to Appendix A. C-SCPI manual pages include the following topics:

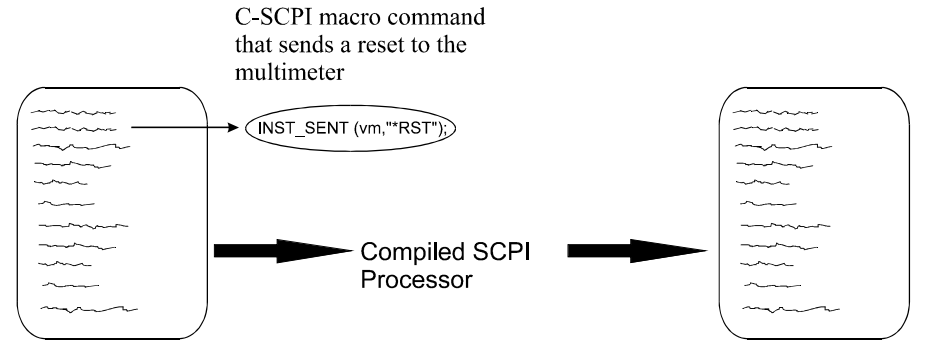
- C-SCPI Macro Commands
- C-SCPI Function Calls
- C-SCPI Preprocessor Command
- Each Supported HP VXI Register-Based Instrument

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Learning About Compiled SCPI

Compiled SCPI (C-SCPI) is a productivity tool designed to aid programmers in achieving high throughput of register-based VXI instrumentation. This is all done while using the easy to understand SCPI language. C-SCPI takes a user's C program with special preprocessor commands and creates C source code.

The user writes the test program in ANSI C language with the C-SCPI preprocessor commands. The preprocessor commands contain specific SCPI commands as arguments. When the C-SCPI preprocessor runs, it traps all C-SCPI commands and replaces them with actual driver calls. If the instrument is message-based or HP-IB (Hewlett-Packard Interface Bus), C-SCPI uses the appropriate HP SICL (Standard Instrument Control Library) function to perform the task. The user then runs the output file through the ANSI C compiler and linker to create the executable code.



The user's standard C code is not affected by the C-SCPI preprocessor. The user determines which ANSI C compiler to use and the code retains its portability. Additionally, standard debugging utilities can be used to debug programs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

What Is VXI?

VXIbus is an open architecture instrument interface for cardcage instrumentation. It was created and is supported by a consortium of manufacturers. Since VXIbus is an open standard, you can have a multi-vendor environment. To find out more about VXIbus, order your own *Feeling Comfortable with VXI* book with HP P/N 5952-3080.

What Is SICL?

HP Standard Instrument Control Library (SICL) provides a modular instrument communications library that works with a variety of computer architectures, I/O interfaces, and operating systems. It uses standard functions to communicate over a wide variety of interfaces. See the HP SICL documentation for additional information.

What Is SCPI?

Standard Commands for Programmable Instruments (SCPI) is an industry standard instrument control language that is supported by a consortium of manufacturers. Using SCPI helps alleviate upgrade and obsolescence problems since instruments understand the same commands regardless of manufacturer. In the U.S, call HP Press at 800-333-0088 to order your *Beginner's Guide to SCPI.*

Who Should Use Compiled SCPI?

This product is intended for software developers with a working knowledge of the C programming language.

Advantages of Using Compiled SCPI

Compiled SCPI allows you to achieve the high throughput of register-based cards with the ease of high-level programming. The following lists the advantages of using C-SCPI:

- -- High throughput of register-based VXI instruments
- -- Easy to understand SCPI commands
- -- Use of industry standards (VXIbus and SCPI)
- -- Supports register-based, message-based, and HP-IB instruments
- -- Can use standard debugging tools
- -- C-SCPI preprocessor has a small command set

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI System

VXI Instruments

See the specific VXI Instrument manual for information on instrument operating instructions and configuration. There is also online documentation provided that contains a quick reference of SCPI commands and detailed information about the card's operation with C-SCPI. See Appendix A later in this guide.

Embedded VXI Controller

The embedded VXI controller is an HP Pentium computer based on the Intel 486™ Pentium processor. In this configuration it acts as a VXI slot 0 system controller and resource manager. The main software items include the following:

- **SICL** includes both the software that allows the controller to act as a resource manager and the SICL I/O library. The resource manager runs automatically at start-up and determines the system configuration. **SICL** is an input/output library that provides communication between the embedded VXI controller and the VXI and/or HP-IB instruments.
- **Lynx Real-Time Operating System** is the program that provides the system control and manages the system's resources.
- **HP Compiled SCPI** is a programming tool that consists of C-SCPI driver libraries and a preprocessor. The C-SCPI preprocessor transforms the C-SCPI programming commands into driver and SICL calls.

More about Compiled SCPI and SICL

Compiled SCPI uses HP SICL calls for communication between devices in your system. C-SCPI allows users to program with the SCPI language to perform instrument tasks. Without C-SCPI, the programmer would have to either use SICL calls to write directly to the instrument's registers or use a command module to interpret the SCPI commands. Writing to the instrument's registers can be very complex and time consuming, and using a command module slows down the program execution speed since the SCPI commands are interpreted at run time instead of compile time as with C-SCPI.

There may be times when you want to program with both C-SCPI and SICL commands. SICL calls are normally used when you have a register-based instrument in your system that is not supported by C-SCPI and it does not have its own driver. In this case you would have to communicate with the instrument's registers or write a driver for this instrument. See Chapter 3 for an example program with embedded SICL calls.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Maximum System Throughput with Compiled SCPI

There are many levels of throughput which you can achieve with HP VXI. The following table shows the relationship between card types, ease of use, and throughput.

Card Type and Programming Language	Throughput and Ease of Use
Message-based card Message commands	Standard throughput Easy to understand commands
Register-based card Command module to interpret SCPI commands	Standard throughput Easy to understand commands
Register-based card Register programming	Best throughput Difficult to program
Register-based card Compiled SCPI to interpret SCPI commands	Better throughput Easy to understand commands
Register-based cards Compiled SCPI running in Overlapped mode	Best throughput Easy to understand commands

C-SCPI provides both high throughput and ease of use. However, to get optimum throughput, you must use the overlapped mode. Using the overlapped mode allows you to execute several commands to different instruments in parallel. See “Overlapped Mode” in Chapter 2 for more information on increasing your throughput with the overlapped mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2

Using Compiled SCPI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using Compiled SCPI

This chapter discusses how to use C-SCPI commands in your C programs and create executable code. Using source code from the source1.cs program, each step is presented and explained. This chapter also describes preprocessor options, interactive functions, and the overlapped mode.

Overview of C-SCPI

Compiled SCPI (C-SCPI) is a productivity tool designed to aid engineers in achieving high throughput with register-based VXI instrumentation. C-SCPI takes a user's C program with special preprocessor commands and creates C source code. The preprocessor commands contain high-level standard SCPI commands as parameters. Before compiling your C program, the C-SCPI commands are translated into C function calls.

The flowchart in Figure 2-1 describes the C-SCPI process.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

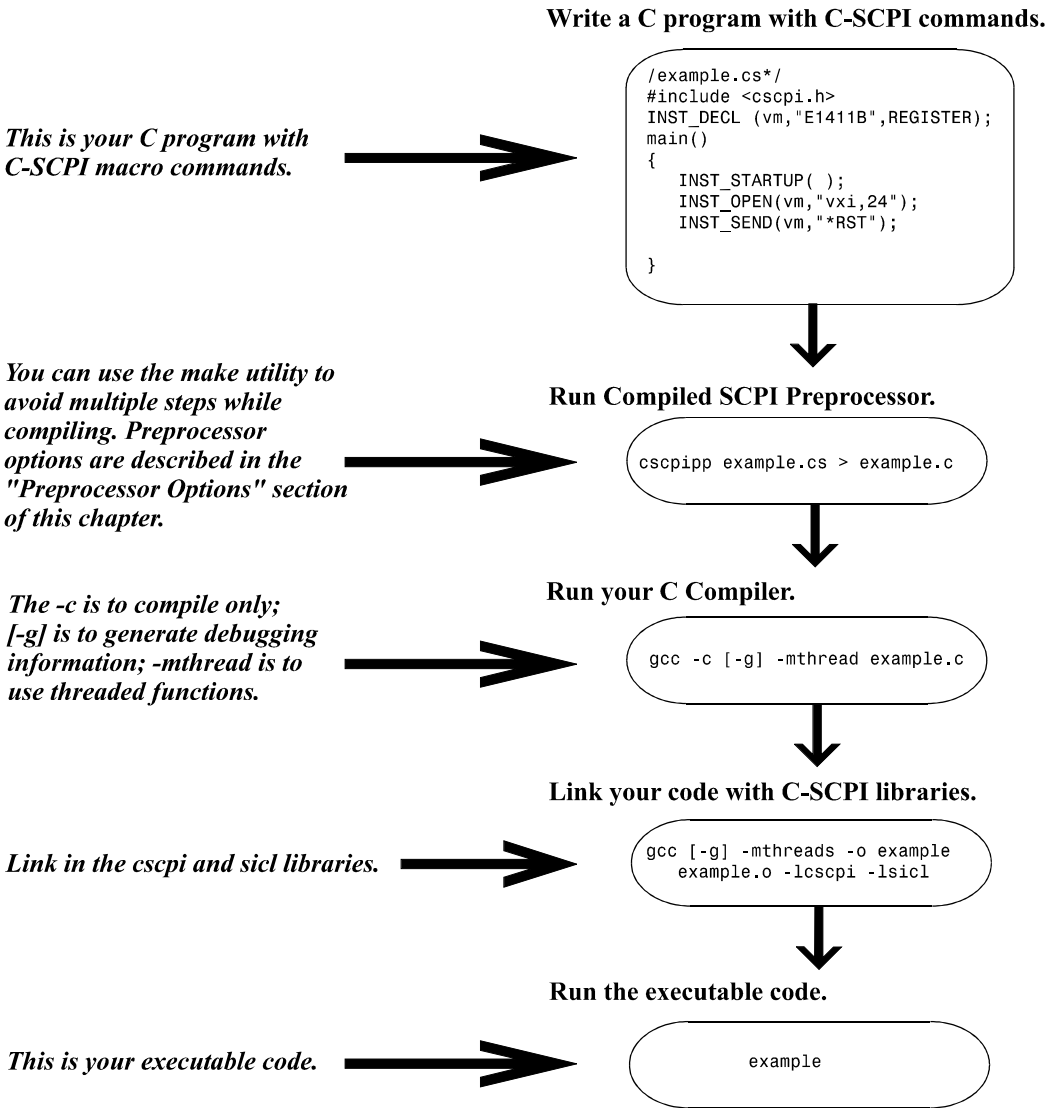


Figure 2-1. Flowchart of C-SCPI Process

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating Source Code

This section describes a C source program with C-SCPI commands. As an experienced C programmer, you are already familiar with using the C language. Now, you can add C-SCPI commands to your program to achieve high throughput from your register-based VXI instruments. You can use the same syntax to operate your HP-IB and message based devices also. The discussion revolves around a short, but reasonably typical C program named `source1.cs`. The `source1.cs` program, like all the examples in this guide, is located in the directory **/usr/hp75000/demos/cscpi**. Once you have copied the sample program into your own directory, you can treat it as you would any C source program. The `source1.cs` program is shown below for the discussion in this chapter.

The `source1.cs` program uses C-SCPI to send SCPI commands to set the voltmeter for DC Voltage, and read the DC Voltage. The voltage is printed using the `printf` function.

```
#include <stdio.h>                  /* include standard C runtime  */
#include <stdlib.h>                 /* header files and the C-SCPI */
#include <cscpi.h>                  /* header file                 */
INST_DECL(vm,"E1411B",REGISTER);    /* global declaration of E1411B*/
                                    /* instrument, register mode   */
void main(){
  float vm_dc;
  float numb1=2.0;

  INST_STARTUP();                   /* initialize instrument       */
                                    /* operating system            */
  INST_OPEN(vm,"vxi,24");           /* open voltmeter at Logical   */
                                    /* Address 24                  */
  if (vm == 0){                     /* check to see if open failed */
    printf("open failed on vm\n");
    printf("cscpi open error number: %d\n", cscpi_open_error);
    exit(1);
  }
  INST_SEND(vm,"CONF:VOLT:DC %f",numb1); /* configure for DC volt   */
  INST_QUERY(vm,"READ?","%f",&vm_dc);    /*query for DC volt reading*/
  printf("DC Voltage is:  %f\n",vm_dc);  /* print the voltage       */
  exit(0);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Defining the C-SCPI Commands

Using C-SCPI commands in your C program allows you to program at a high-level instead of performing register reads and writes. Refer to the "Compiled SCPI Command Reference" section later in this guide for a complete list and description of the C-SCPI commands. The following list describes the C-SCPI commands used in the `source1.cs` program:

- **Header File** - provides function prototype and variable typing information that will be used by C-SCPI commands.

```
#include <cscpi.h>
```

- **Instrument Declaration** - makes your instrument declaration using the `INST_DECL` command. This declares *vm* for you as type `INST`. `INST_DECL` may be a global or local declaration.

```
INST_DECL(vm,"E1411",REGISTER
```

- **Instrument Initialization** - initializes your instruments using C-SCPI commands. `INST_STARTUP` starts the instrument register-based operating system. It must be executed before C-SCPI commands, except `INST_DECL`. `INST_OPEN` initializes your instrument and instrument driver. It must be executed after `INST_STARTUP` and before other instrument commands.

```
INST_STARTUP( );
INST_OPEN (vm,"vxi,24");
```

- **Instrument Programming Commands** - send commands to your instruments, and query for the results using standard SCPI commands with the most commonly used C-SCPI commands, `INST_SEND` and `INST_QUERY`.

```
INST_SEND(vm,"CONF:VOLT:DC %f",numb1);
INST_QUERY(vm,"READ?","%f",&vm_dc);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Running the C-SCPI Preprocessor

This section provides a guide to running the C-SCPI preprocessor. Figure 2-2 displays the preprocessor process.

To run the C-SCPI preprocessor, type the following at the command prompt:

```
cscpipp  source1.cs > source1.c
```

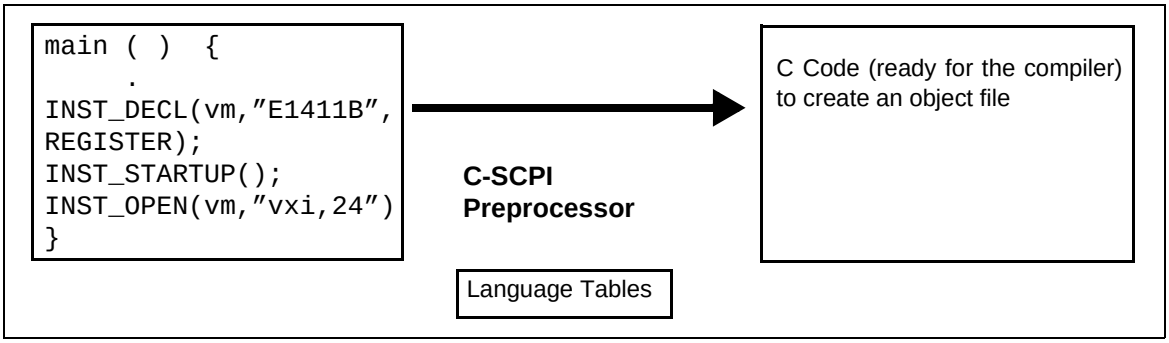


Figure 2-2. C-SCPI Preprocessor Process

C-SCPI commands are preprocessed and translated into ANSI C function calls. Figure 2-3 displays a sample of the ANSI C source code (`source1.cs`) with C-SCPI commands, and the translation of the C-SCPI commands (`source1.c`) into ANSI C function calls after running the preprocessor. The resulting ANSI C file contains code generated by the C-SCPI preprocessor, and is not intended to be modified by the user. However, line mapping information in the resulting ANSI C file will cause the C compiler to reference the original source file (e.g., `source1.cs`) for any errors or warnings that you may receive.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

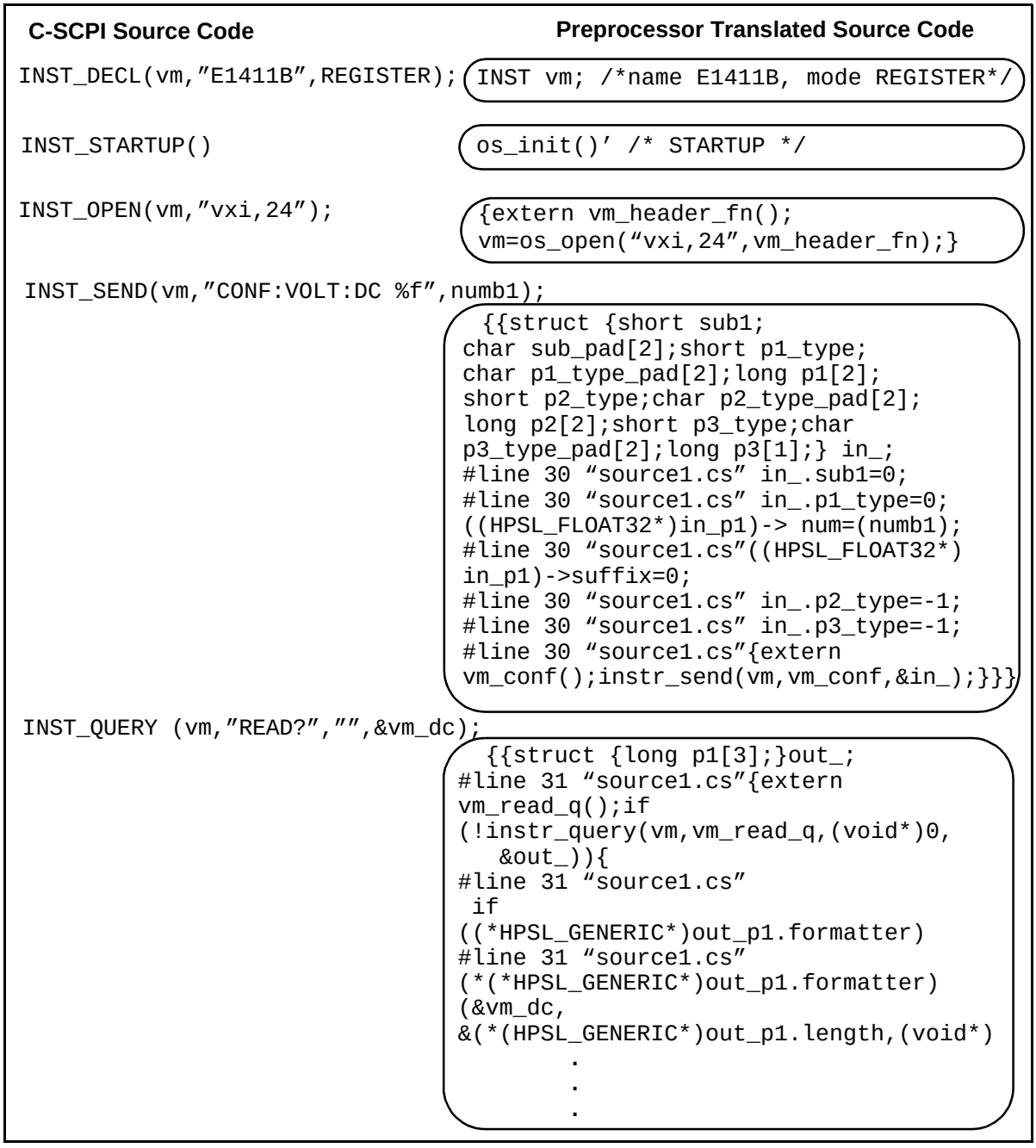


Figure 2-3. C-CSPI Source Code and Preprocessor Translated Code

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiling/Linking your Code

This section explains how to compile and link your source code to create an executable file. When creating your executable file, you can compile and link in one or two steps as described below. For additional information on compiling and linking programs, refer to your C compiler's manuals. Figure 2-4 displays the compile and link process.

- **Compiling**. During this step the C compiler converts the C source file(s) to an *object file*. An *object file* contains binary code, but it is not in executable form. To compile your program, type the following at the command prompt:

```
gcc -c [-g] -mthreads source1.c
```

Comments

-- The **`-c option`** creates an object file called source1.o

-- The **`[-g] option`** indicates to the compiler that the source should be compiled with information so a debugger may be used (this is optional)

-- The **`-mthreads option`** causes the multi-threaded versions of standard functions (like `malloc` and `printf`) to be used rather than the single-threaded versions

-- The **`-I option`** searches the specified directory path for additional include files

-- For troubleshooting Compile errors, see Chapter 4, "Troubleshooting Compiled SCPI"

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

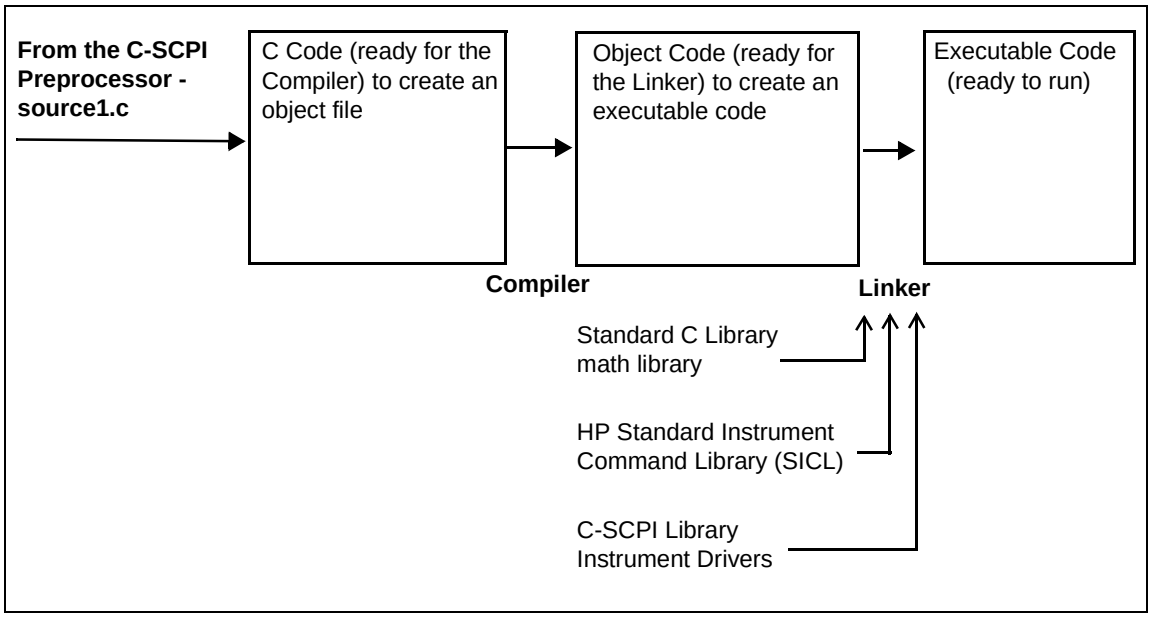


Figure 2-4. Compile and Link Process

- **Linking**. During this step the linker takes the object file created during compilation, combined with standard libraries, plus other object files and libraries you specify, and creates an executable file.
  To link your program, type the following at the command prompt:

```
gcc [-g] -mthreads -o source1 source1.o
   -lcscpi -lsicl
```

Comments

-- The **cscpi** library must precede the **sicl** library when linking your program

-- The **`-o option`** creates an executable file called source1 rather than the default a.out

-- The **`-l option`** is used to link in the Compiled SCPI library and the SICL library

-- For troubleshooting Link errors, see Chapter 4, “Troubleshooting Compiled SCPI”

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- **Compiling/Linking in One Step**. You can also compile and link your program in one step. To compile/link in one step, type the following at the command prompt:

```
gcc [-g] -mthreads -o source1 source1.c
  -lcscpi -lsicl
```

Comments

-- This will create the executable file, `source1`

-- For troubleshooting compile/link errors, see Chapter 4, "Troubleshooting Compiled SCPI"

Using Libraries

A **library** is a set of commonly used functions that have been gathered into one place. The library functions are already assembled or compiled to object code. They are referenced by the linker to create an executable file.

In addition to the C library, you need the following libraries to compile your C-SCPI programs:

cscpi The Compiled SCPI Library provides access to the instrument drivers for all SCPI commands. Additionally, it provides the instrument environment necessary to interface between your computer's operating system and the instrument drivers.

sicl The Standard Instrument Control Library provides a modular instrument communications library that works with a variety of computer architectures, I/O interfaces, and operating systems. By using this library, C-SCPI remains independent of Input/Output machine specific information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Executing Your Program

To execute your program, type the following at the command prompt:

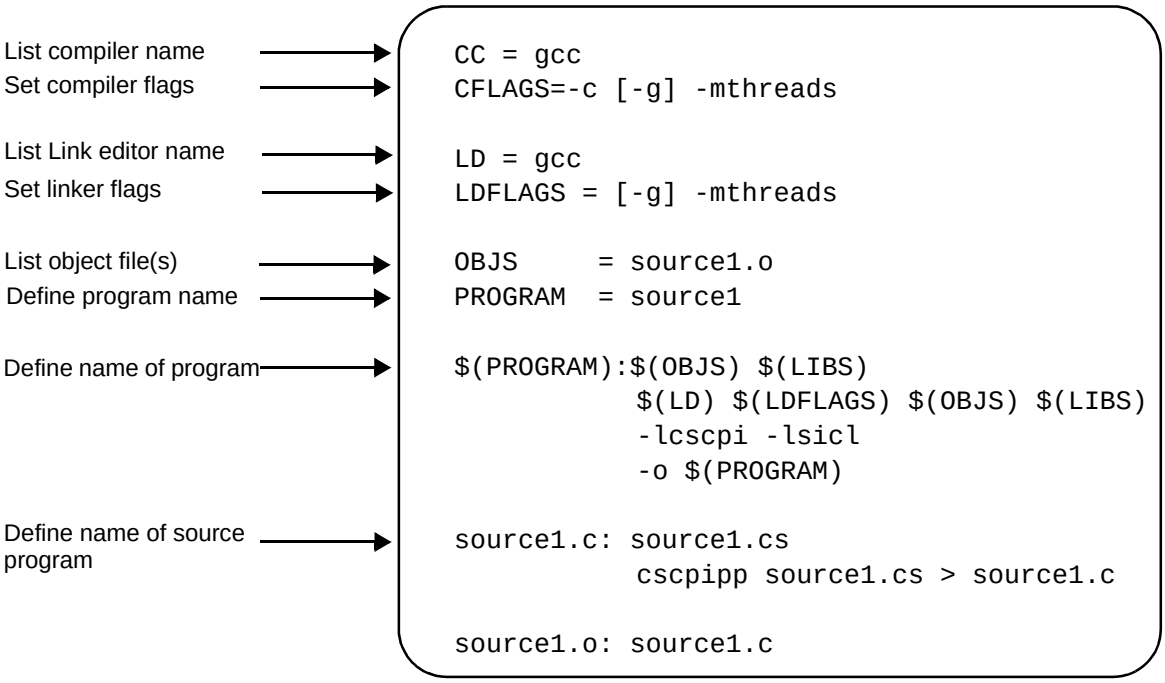
```
source1
```

The result of executing this program is as follows:

```
DC Voltage is: 1.86321
```

Using Makefiles

Using make, you can create a makefile to ease compiling and linking programs into one executable file. For more information on how to use the *make* command, refer to the Lynx manuals. The following is an example of a simple makefile to compile/link the C-SCPI program `source1.cs`.



Note a TAB must precede `$(LD)` and `cscpipp` lines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

This is a very simple makefile. If your system has more than one C-SCPI program, then you may want to use the `SUFFIX` option of makefile. This option tells the *make* command how to convert your C-SCPI programs (`.cs` files) into C programs (`.c` files). There are a number of ways to do this. The example below is one of those ways. It tells *make* how to convert your `.cs` files into `.c` files, without mentioning any specific program name.

Add the following two lines to your makefile to convert C-SCPI programs (`.cs` files) into C programs (`.c` files):

```
SUFFIXES:   .cs
cs.c:;      cscipp $<> $*.c
```

Note A TAB must precede `cscipp $<> $*.c`

By putting these two lines at the beginning of your makefile, you would not need the lines that “Define name of source program” in the previous example. This approach uses *suffix rules* to accomplish the C-SCPI preprocessing step. The makefile in the examples directory uses this approach to compile C-SCPI example code.

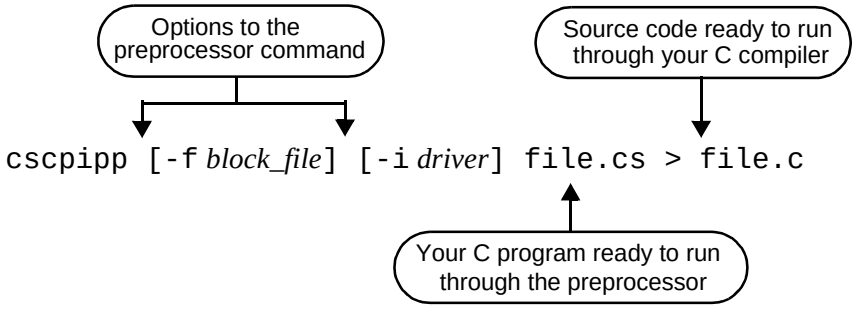
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Preprocessor Options

This section contains a short description of the C-SCPI preprocessor command and its options.

The Compiled SCPI Preprocessor Command

When using C-SCPI, the preprocessor command is as follows:



The text in brackets ([ ]) are options discussed in the next section.

Storing Block Data in a Separate File (The -f Option)

When using SCPI commands that require block data to be sent to an instrument, the `-f option` can be used with the C-SCPI preprocessor to store this block data in a file outside the C program. This helps keep the size of the C program down, the speed of the compilation up, and allows for a smaller executable file. This option is usually only needed if you have a lot of block data.

Preprocessor Command with -f Option

The C-SCPI preprocessor command with the `-f option` can be used as follows:

```
cscpipp -f example.dat example.cs > example.c
```

Where **`-f example.dat example.cs`** specifies that all block data in the program called **`example.cs`** is to be stored in the file call **`example.dat`** when the preprocessor runs. The resulting C program is called **`example.c`**.

See Chapter 3, “Programming with Compiled SCPI” for a more detailed example program using the `-f option`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example Program

The following program segment contains a SCPI command that sends block data to an instrument. When the `-f option` is used with the C-SCPI preprocessor (as shown in the previous section), the data will be stored in the file specified in the command line.

Additionally, in order to get the data when you run your program, you must open the file that contains the block data and assign it to the `cscpi_datafile` pointer (shown below).

`example.cs`

```
FILE *cscpi_datafile;/*assign pointer for data retrieval*/
  .                          <-- assign cscpi_datafile pointer
  .
  .
main()
{
  .
  .
  .
  /*open file and assign to cscpi_datafile*/

  cscpi_datafile = fopen ("example.dat", "rb");   <-- open file and assign to pointer
  .
  .
  .
  INST_SEND (digio, "SOURCE:DIGITAL:TRACE:DATA first_block,
    #210ABCDEFGHIJ");
  .
  .
}
```

The `INST_SEND` command has a SCPI command that sends block data to the digio. When the C-SCPI preprocessor is run with the `-f option`, the data that is stored in the `example.dat` file. See the *HP E1330B Digital I/O User's Manual* for more information on the SCPI command.

In order to read the block data when the program is executed, the file containing the block data must be opened and assigned to the `cscpi_datafile` pointer. Therefore, in the example above, the file called `example.dat` is opened and assigned to the pointer `cscpi_datafile`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note C-SCPI requires the name of the pointer to the data file to be `cscpi_datafile`. The name of the file, however, is defined by the user.

Using SCPI Only Files (The -i Option)

The `-i option` takes a file that contains only SCPI commands and converts it into a C function. The C-SCPI preprocessor treats each SCPI command in the file like an `INST_SEND`.

When using the C-SCPI `-i option`, the following restrictions apply:

-- All SCPI commands must be for the same instrument.

-- SCPI query commands are NOT allowed.

-- Any text that is not a SCPI command is NOT allowed (for example, comments).

Preprocessor Command with -i Option

The C-SCPI preprocessor command with the `-i option` can be used as follows:

```
cscpipp -i E1411B scpifile.cs > scpifile.c
```

Where **`-i E1411B scpifile.cs`** specifies that the file **`scpifile.cs`** contains only SCPI commands for a multimeter. Each command is treated as a C-SCPI `INST_SEND`. The resulting C program is called **`scpifile.c`**.

To create your executable code, you must run the C-SCPI preprocessor on each file, compile each file, and then link the files together. An example follows:

- Run the C-SCPI Preprocessor:

```
cscpipp -i E1411B scpifile.cs > scpifile.c
cscpipp example.cs > example.c
```

- Compile each file:

```
gcc -c [-g] -mthreads example.c
gcc -c [-g] -mthreads scpifile.c
```

- Link the files:

```
gcc [-g] -mthreads -o example example.o scpifile.o
   -lcscpi -lsicl
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example Programs

This section shows two program segments. The first is a file segment that has SCPI commands for the HP E1411B Multimeter. The second program segment shows a main program that uses the first file which contains only SCPI commands. The C-SCPI `-i option` is used with the first file to convert it into a C function of instrument sends.

The following program segment shows a file containing strictly SCPI commands for the HP E1411B Multimeter. When this file runs with the `-i option`, a function called `scpifile` is generated (same as the file name but without the extension).

`scpifile.cs`

```
*RST
*CLR
FUNC VOLT:AC
VOLT:RANGE 8
  .
  .
  .
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

This program segment calls the function generated by the C-SCPI preprocessor with the `-i option`. The instrument *driver* name that was specified with the `-i option` is used as the pass parameter when calling the function.

`example.cs`

```
#include <stdio.h>
#include <cscpi.h>

extern void scpifile(INST_PARAM(vm, “E1411B”, REGISTER));
  .
  .
  .
main()
{
  INST_DECL (vm, “E1411B”, REGISTER);
  INST_STARTUP ();
  INST_OPEN (vm, “vxi,24”);
     .
     .
     .
  scpifile (vm);
     .
     .
     .
}
```

Each file must be run through the C-SCPI preprocessor, compiled, and then linked. When the first file, `scpifile.cs`, is run through the C-SCPI preprocessor with the `-i option`, the `scpifile` function is generated. When the main program calls the function, the instrument *driver* name specified when the preprocessor was run is used.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interactive Functions

The interactive functions allow you to specify your SCPI command at run time instead of in your source code. With other C-SCPI commands you have to specify the SCPI command in the argument list. For example, `INST_SEND(id, "*RST")` contains the SCPI command `*RST`. With the `INST_SEND` command, the SCPI command must be in your program before the C-SCPI preprocessor runs. With the interactive functions, however, you can prompt the user to enter the SCPI command at run time.

Advantages of Using the Interactive Functions

Several advantages of using the interactive functions include the following:

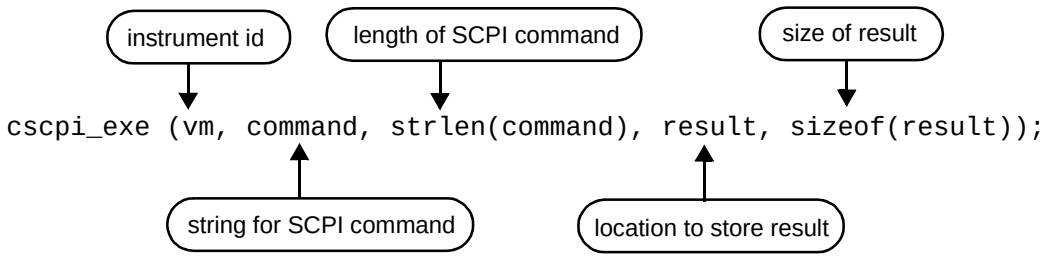
- You can enter the SCPI command at run time instead of in your source code.
- You can use the interactive functions as a means of debugging your SCPI commands.

The only disadvantage of using the interactive functions is that error checking and parsing of SCPI commands is not done until run time (instead of checked when the C-SCPI preprocessor runs). This, in turn, slows down the program execution speed.

Using Interactive Functions

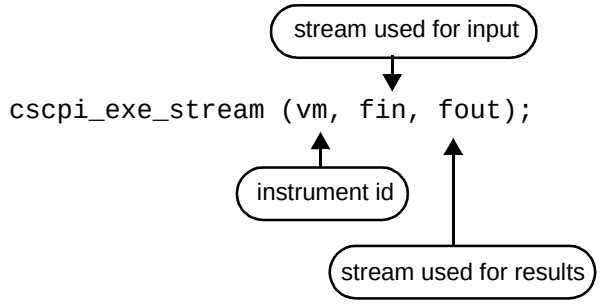
There are three interactive functions available with C-SCPI:

The `cscpi_exe` function uses a string variable for input to the function. The results are stored in the address specified. With this function you have to specify the length of the SCPI command string and the size of the location used to store the results. The following illustrates the function call:

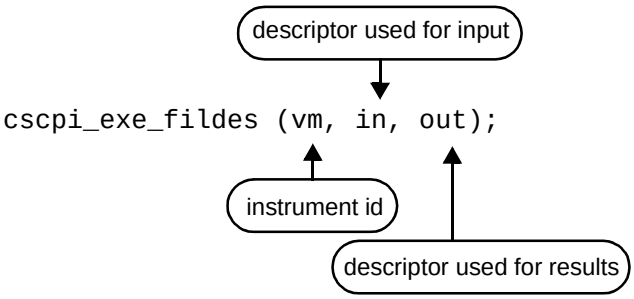


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The `cscpi_exe_stream` function allows you to use streams as the file input and output. The following illustrates the function call:



The `cscpi_exe_fildes` function uses a file descriptor as file input and output. The following illustrates the function call:



See Chapter 5, “Compiled SCPI Command Reference” for more information on the C-SCPI interactive functions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming with Interactive Functions

Since the interactive functions allow you to enter the SCPI commands at run time, you can write your program to prompt the user for a SCPI command at run time. An example program using the `cscpi_exe` function is described in this section. This example is stored in the **/usr/hp75000/demos/cscpi** directory. See Chapter 5, "Compiled SCPI Command Reference" for examples of the other two interactive functions.

Equipment Needed

-- HP Embedded VXI Controller
-- HP VXI C-Size Mainframe
-- HP E1411B Digital Multimeter

Program Description

This program sets up and initializes the HP E1411B Multimeter. It prompts the user for an address and loops prompting for SCPI commands. The SCPI commands are parsed by the Controller and executed by the HP E1411B Multimeter. When a null string is entered, the loop is exited.

This program demonstrates how you can use the C-SCPI execute function in an interactive mode. See Chapter 5, "Compiled SCPI Command Reference" for more details on the C-SCPI execute call.

Program Listing

```
/*example6.cs*/
/*This is a C-SCPI example of using the interactive mode. The  */
/*program is written to prompt the user for the SCPI command.  */
/*The command is then executed using the cscpi_exe function.   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cscpi.h>

#define LENGTH 1000  /*maximum length of SCPI command & result*/

INST_DECL (vm, "E1411B", REGISTER);/*declaration for voltmeter*/

main()
{
  char command [LENGTH];   /*string variable for SCPI command*/
  char result [LENGTH];    /*string variable for result*/
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  /*prompt user to enter logical address of multimeter*/
  puts (“Enter the logical address of the vm, for example,”
    “ vxi,24: \n”);
   gets (command);

  INST_STARTUP ();              /*start operating system*/
  INST_OPEN (vm, command);      /*initialize multimeter*/

  /*if 0 returned, open failed and print message*/
  if (!vm)
  {
    fprintf (stderr, “vm at %s failed to open\n”,command);
    exit (1);
}

/*loop to enter SCPI commands*/

for (;;)
  {
    printf (“Enter a SCPI command for the multimeter. Hit return “
      ”to exit\n”);
    while (!gets(command)) /*loop until get a nonzero size  */
      ;                    /*gets may terminate on interrupt*/
    if (!*command)         /*caused by the command*/
      break;
    result[0] = 0;         /*clear result string*/

    /*C-SCPI call to execute the SCPI command*/
    cscpi_exe (vm, command, strlen(command),result, sizeof(result));

    /*if you have a result, print it*/
    if (result[0])
      printf (“Result :  %s”, result);
  }
  printf (“DONE\n”);
  exit (0);
}
```

Note The char array returned for result includes a new line at the end of the array. Therefore, you do not have to include a new line when printing the results.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Overlapped Mode

The overlapped mode is a mode of operation which allows you to overlap commands addressed to different instruments so that they are executed in parallel. The default mode (overlapped off) does sequential programming, that is, one command is started after the previous command has finished. With the overlapped mode on, however, you can take advantage of long setup times and allow commands to overlap, thereby, achieving higher throughput.

Turning overlapped mode on and off only applies to register-based instruments. Message-based instruments automatically operate in overlapped mode because they have their own processor.

While using the overlapped mode, certain commands can be executed in parallel. When the command is executed, it begins the instrument function and completes it some time later with an interrupt service routine. Meanwhile, other commands may begin execution without having to wait for the instrument function to complete.

The overlapped mode can be turned on or off in your C program. By default this mode is off. See “Using the Overlapped Mode” on page 40 for a discussion on the C-SCPI function that turns the overlapped mode on or off.

Determining if you Should use the Overlapped Mode

The overlapped mode can provide an increase in system throughput. Use overlapped mode for either of the following conditions:

-- Your configuration requires commands to be executed in parallel.

OR

-- You need an increase in system throughput.

See “Controlling Overlapped Execution” on page 43 for a description of programming techniques you can use.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Configurations Requiring Overlapped Mode

Overlapped mode must be used when you have a configuration that requires SCPI commands for two or more instruments to be executed in parallel. An example of this configuration might be if you want to scan several channels and make measurements on each channel, where the instruments are set up as two separate instruments (not a scanning voltmeter). In this configuration, a switch should close and wait for a voltmeter complete (indicating that the voltmeter made its measurement). Then the switch should close the next channel and the process should repeat. However, since the switch's `INIT` command is executed later in the program, it will never be executed without overlapped mode ON. This is because the voltmeter's measurement command is never finished.

The following program segment shows this configuration with overlapped mode OFF:

```
INST_SEND (vm, "TRIG:SOUR EXT");
INST_SEND (vm, "CONF:VOLT:DC 0.825,MAX");
INST_SEND (sw, "TRIG:SOUR EXT");
INST_SEND (sw, "ROUT:SCAN (@100:103)");
INST_QUERY (vm, "READ?" "", &result);
  .
  .
  .
INST_SEND (sw, "INIT");
```

Notice that the switch `INIT` is after the voltmeter `READ`. Since the voltmeter requires an external trigger, the `READ` command hangs because the `INIT` command is what causes the voltmeter trigger to occur. However, the `INIT` command is never reached.

Now, if you turn overlapped mode ON, this configuration should work.

Throughput and the Overlapped Mode

Since the overlapped mode allows several commands to execute in parallel, the system throughput can be higher. This section describes why the throughput can be higher with overlapped mode ON and compares two C-SCPI programming segments. One program executes the commands sequentially, and the other program executes the commands while in the overlapped mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comparing Two Programs

Both programs shown in Figure 2-5 are making the same measurement. The second program is in overlapped mode and the order of command execution is different. See “Programming for Efficiency” on page 45 for a description of how to determine the order of your commands while using the overlapped mode.

The first program in Figure 2-5 uses sequential commands and sends the commands in a logical order for execution. The controller waits until each command is done before beginning the next command. These commands are executed sequentially.

The second program in Figure 2-5 uses the overlapped mode and sends all of the setup commands first. If these commands are overlapping commands and sent to different instruments, they are executed in parallel. Once the setup commands are sent, `*OPC?` commands are sent to make sure all of the SCPI commands are done before taking a measurement.

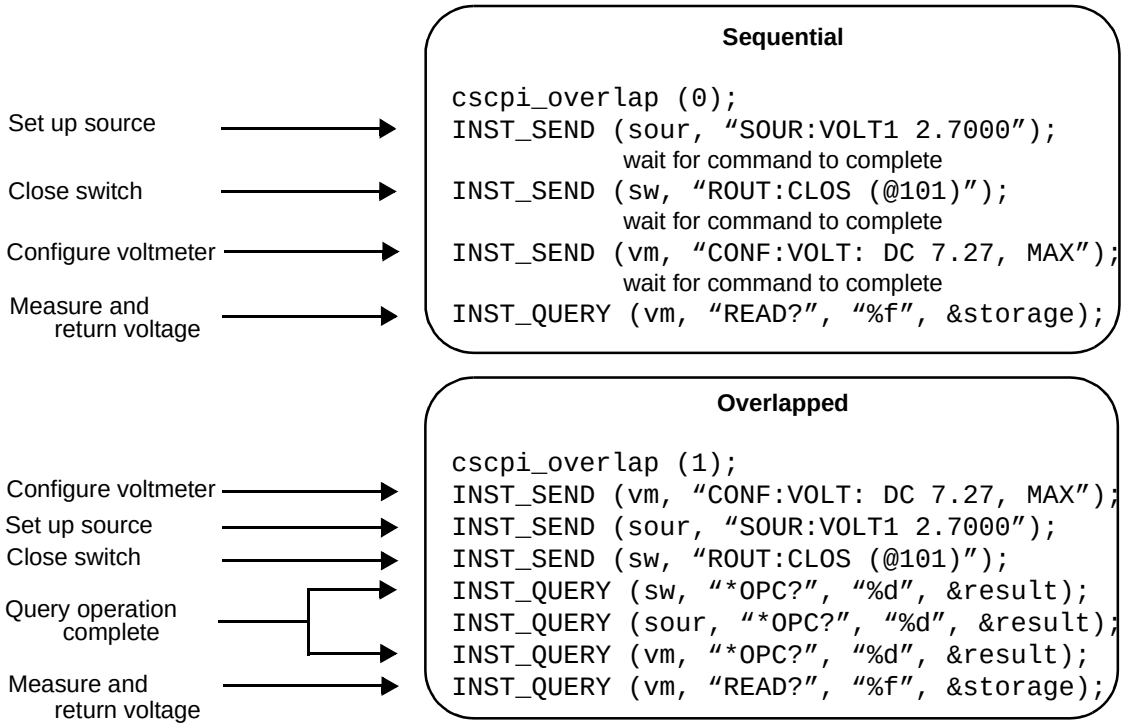


Figure 2-5. Programming Code, Overlapped vs. Non-Overlapped

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

While the program segments are listed in Figure 2-5, Figure 2-6 illustrates the time difference between the sequential and overlapped programs. In the sequential program version, with overlapped mode OFF, each command will wait for completion before executing the next line. With `MESSAGE` configurations, however, you must include a `*OPC?` command to ensure the commands are completed.

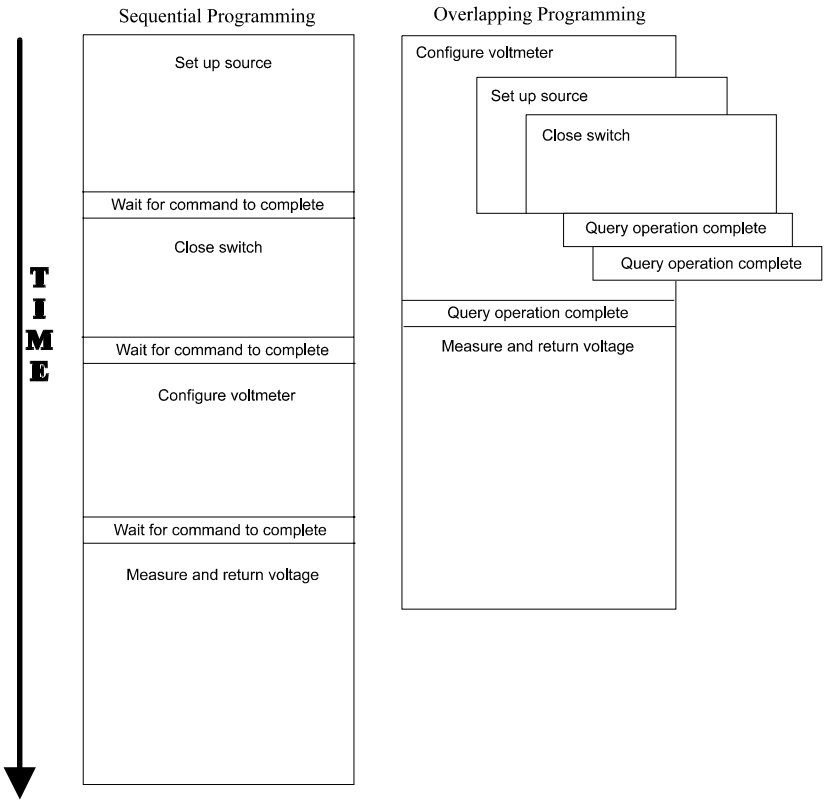


Figure 2-6. Time for Sequential and Overlapped Programs

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Overlapped Command

An overlapped command is a command that allows other commands to be executed at the same time. Some commands can be overlapped, and some can not. Some commands will not allow any other command to be executed after the command has started. Therefore, we have two types of commands:

- A **NON-OVERLAPPING** command is a command that does not allow any other command to be executed after the command has begun. The non-overlapping command must complete execution before the next command can start.
- An **OVERLAPPING** command is a command that allows other commands (overlapping or non-overlapping) to other instruments to be executed at the same time. An overlapping command continues execution when an interrupt from the hardware card is delivered to the test program. Overlapping commands ONLY overlap when the overlapped mode in turned ON.

To find out which commands are overlapping and which are non-overlapping, see Appendix A, “Online Documentation” for more information.

Using the Overlapped Mode

To use the overlapped mode you simply place a C-SCPI function call in your C program to turn the mode ON or OFF. Default mode is overlapped OFF. There are two C-SCPI functions that are associated with the overlapped mode:

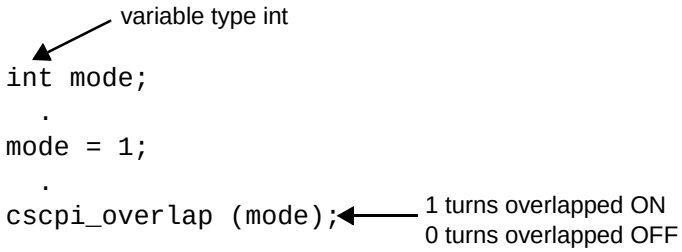
- `cscpi_overlap`
- `cscpi_get_overlap`

Each of these C-SCPI functions is discussed in the sections following. Make sure you read the “Controlling Overlapped Execution” on page 43 before attempting to use the overlapped mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Turning Overlapped Mode ON or OFF

The `cscpi_overlap` function turns the overlapped mode ON or OFF. By default, this mode is turned OFF. If you use a nonzero integer as the parameter, the function turns overlapped mode ON. A `0` turns overlapped mode OFF. The parameter is of type `int`. The following program segment shows how this function can be used in your C program:



The `1` turns overlap ON (`0` would indicate OFF). See Chapter 5 for more information on the `cscpi_overlap` call.

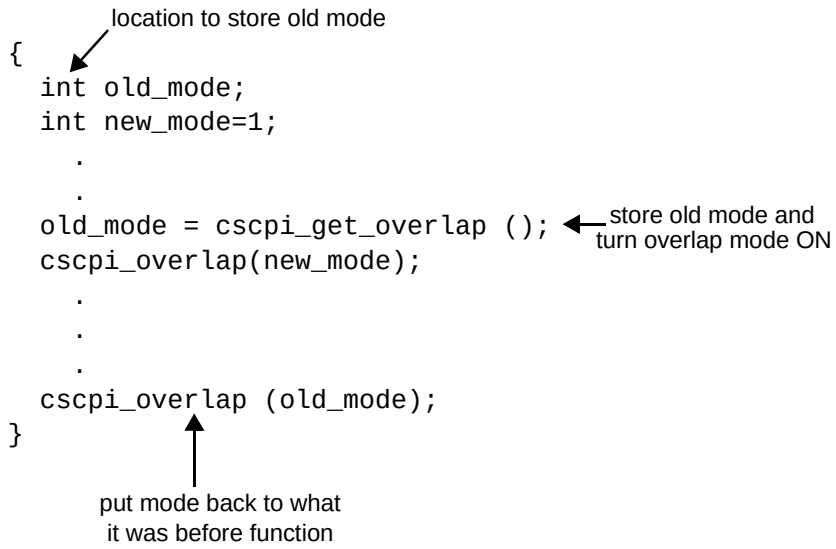
Note

Turning overlapped mode OFF does not guarantee that interrupts from incomplete commands are not generated. If overlapping commands were executed while overlapped mode was ON, additional interrupts can be generated if those commands have not completed yet.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Determining if Overlapped Mode is ON or OFF

The `cscpi_get_overlap` function returns an integer that tells if the overlapped mode is ON or OFF. If a `1` is returned, overlapped mode is ON. A `0` indicates that overlapped mode is OFF. The following program segment shows how you can store the current status of overlapped mode, turn overlapped ON, perform a function, and then return overlapped mode to the same status it was before the function:



The returned mode tells if overlapped mode is ON or OFF. This mode is stored and the overlapped mode is turned ON. Once the function is done, overlapped mode is put back to whatever it was before the function began. See Chapter 5 for more information on the `cscpi_get_overlap` call.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controlling Overlapped Execution

Most C-SCPI instrument drivers use interrupts to perform their tasks. When overlapped mode is on, these interrupts may interrupt the execution of your test program. The following example illustrates how interrupts could occur:

Configure Voltmeter →
Set up source →
Close switch →

```
INST_SEND (vm, “CONF:VOLT: DC 7.27, MAX”);
INST_SEND (sour, “SOUR:VOLT1 2.7000”);
INST_SEND (sw, “ROUT:CLOS (@101)”);
    YOUR CODE HERE
  .
  .
```

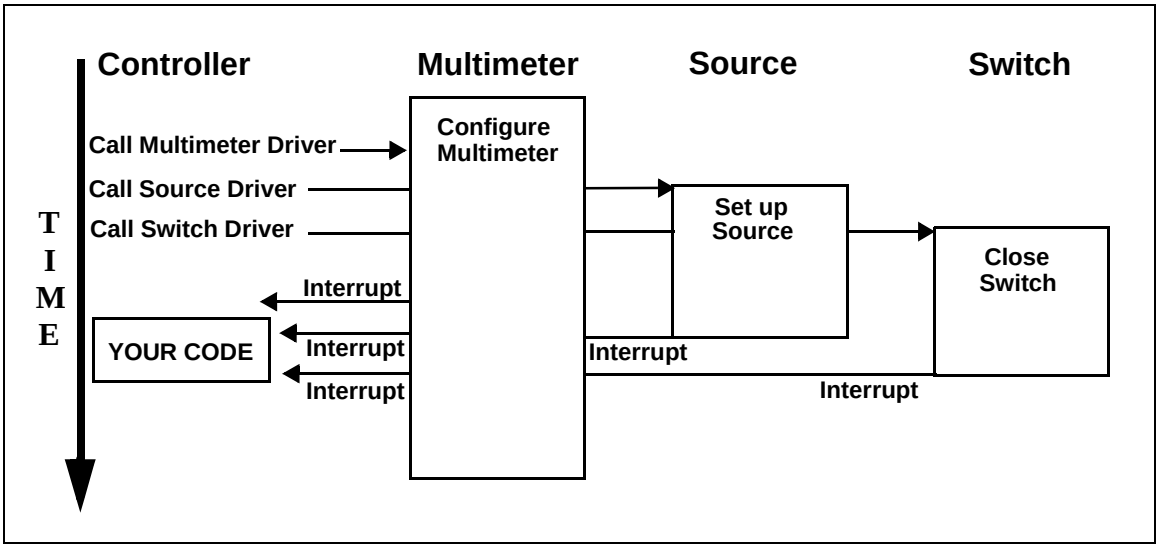


Figure 2-7. When an Interrupt Occurs

If you have any problems with your code because of interrupts, we recommend you do one of the following:

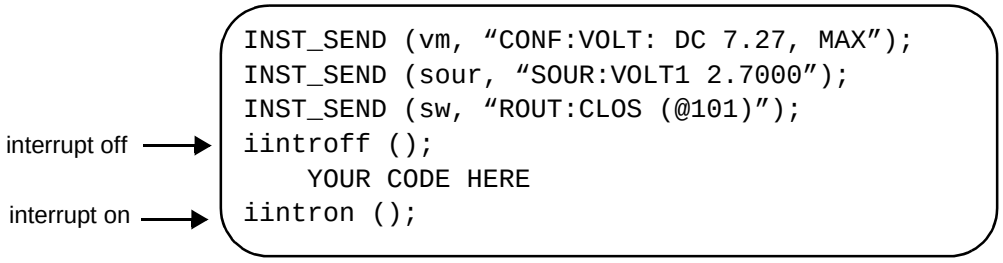
-- Leave overlapped mode in its default mode (OFF).
-- Use HP SICL function calls to temporarily block interrupts.
-- Use `*OPC?` to finish overlapping commands.

The last two items are discussed in the subsections that follow.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Use the HP SICL Function Calls to Temporarily Block Interrupts

You can place HP SICL calls around C code calls to temporarily block interrupts. Once you complete your code, you can re-enable interrupts. The following example places SICL function calls before and after the user code:




```
INST_SEND (vm, “CONF:VOLT: DC 7.27, MAX”);
INST_SEND (sour, “SOUR:VOLT1 2.7000”);
INST_SEND (sw, “ROUT:CLOS (@101)”);
iintroff ();
    YOUR CODE HERE
iintron ();
```

See the HP SICL documentation for additional information on these commands.

Use *OPC? to Finish SCPI Commands

If you are in the overlapped mode you can use `*OPC?` after the SCPI command to ensure the command has finished execution. The following example uses `*OPC?` to ensure SCPI commands are done:

```
int result;

INST_SEND (vm, “CONF:VOLT: DC 7.27, MAX”);
INST_SEND (sour, “SOUR:VOLT1 2.7000”);
INST_SEND (sw, “ROUT:CLOS (@101)”);
INST_QUERY (sw, “*OPC?”, “%d”, &result);
INST_QUERY (sour, “*OPC?”,”%d”, &result);
INST_QUERY (vm, “*OPC?”, “%d”, &result);
printf (“This is a test, send %s\n”, text);
  .
  .
```

Notice that the write to terminal is after all of the `*OPC?` commands.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming for Efficiency

Once you have decided to use the overlapped mode, you can make your program more efficient by doing the following:

-- Change the order of the SCPI commands to increase your system throughput.
-- Use `*OPC?` commands to ensure commands have completed.

Each of these items is discussed in the subsections following.

Determining the Order of your Overlapping Commands

To obtain optimum throughput, you must write your program so that commands are executed in a particular order. Use the following procedure to tune your program to get the best throughput:

1. Determine if the command is overlapping or non-overlapping. See Appendix A, “Online Documentation” for the specific instrument.
2. Determine which overlapping commands take the most time. See the “System Characteristics” section of the *HP 75000 Family of VXI Products* catalog for a list of card execution times.
3. Program the overlapping commands that take the most time first.

Note

You will only gain throughput by overlapping commands to different instruments.

Suppose you have three commands for three different instruments and one is non-overlapping and the other two are overlapping. Send the overlapping command that takes the most time first. Then send the other overlapping command and finally the non-overlapping command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using the *OPC? Commands

In order to guarantee that commands have finished before you try to make a measurement, use the operation complete query command (`*OPC?`). By using this command you can avoid incorrect data. The following program segment is an example of using `*OPC?`.

```
int result;
INST_SEND (sw1, “CLOSE (@101)”);
INST_QUERY (sw1, “*OPC?”, “%d”, &result);
```

Note

The `*WAI` command takes slightly less time than the `*OPC?` command. However, `*WAI` does not have the desired effect on `MESSAGE` instruments. Therefore, `*OPC?` is recommended for portability.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3

Programming with Compiled SCPI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming with Compiled SCPI

This chapter provides some C-SCPI programming examples to help you write your own programs with C-SCPI. Each program contains a list of equipment needed, program description, and a program listing. You can copy any of these examples into your own directory and make changes. The examples in this chapter are stored in the **/usr/hp75000/demos/cscpi** directory.
This chapter contains the following sections:

-- Looking at an Example System Configuration
-- Providing an Error Routine
-- Programming with a Scanning Multimeter
-- Programming with an External File
-- Programming with a C-SCPI Parameter List
-- Storing Block Data in a Separate File (the -f Option)
-- Using C-SCPI in the Interactive Mode
-- Triggering with the HP Pentium Controller

Looking at an Example System Configuration

The configuration shown in this section can be used with the programming examples in this chapter. See the *C-Size VXIbus Systems Installation and Getting Started Guide* and the individual instrument user's manuals for wiring and connection information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Embedded VXI Controller

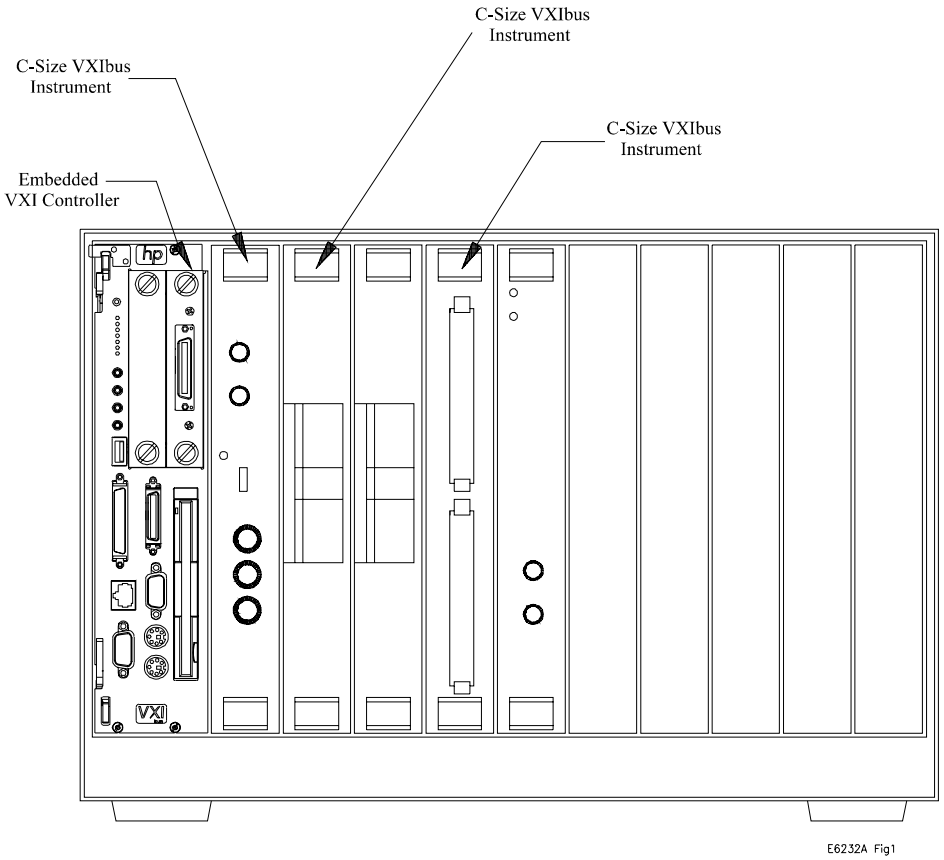


Figure 3-1. Embedded Controller Example Configuration

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Providing an Error Routine

C-SCPI allows for error trapping of instrument run-time errors. Every time a run-time error is put into an instrument's error queue, the `cscpi_error` function is called. The `cscpi_error` function shipped does nothing. You can, however, write your own `cscpi_error` function and link it into your main program. You can also use the example error routine that's been provided in the **/usr/hp75000/demos/cscpi** directory. To use the example error routine, do the following:

1. Copy the `cscpi_error.c` routine into your own working directory.

2. Compile both your file and the `cscpi_error.c` file.

3. Link the files. For example:

```
gcc [-g] -mthreads -o example example.o
    cscpi_error.o -lcscpi -lsicl
```

Now, when you execute your program and a run-time error is put into any instrument's error queue, the new `cscpi_error` function will be called.

Program Listing

```
/*cscpi_error.c*/
/*This routine provides the SCPI error routine for run time errors.*/
/*Errors in the instrument's error queue are reported.*/

#include <cscpi.h>
void cscpi_error (INST sicl_inst, int error_number)
{
  char string[20]="SYST:ERR?";
  char result[255];

 cscpi_exe(sicl_inst,string,strlen(string),result,sizeof(result));

  printf ("ERROR:  %s",result);
  exit(1);
}
```

See "Trapping Errors with cscpi_error" in Chapter 4 for more information on the `cscpi_error` routine.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming with a Scanning Multimeter

This programming example shows how you can use C-SCPI macro commands to set up a measurement for a scanning voltmeter.
See Chapter 5 for more information on these commands.

Equipment Needed

-- Embedded VXI Controller
-- HP VXI Mainframe
-- HP E1326B or HP E1411B Multimeter
-- HP E1345A Relay Multiplexer

Program Description

This program uses C-SCPI commands to set up a measurement for the Multimeter and the Relay Multiplexer to make a scanning measurement.

Program Listing

```
/*example1.cs*/
/*This programming example uses the C-SCPI commands to make a    */
/*measurement for a scanning voltmeter.                          */

#include <stdio.h>
#include <math.h>
#include <cscpi.h>                /*Needed for Preprocessor      */
                                  /* commands.                   */
int test1(void);                  /*function prototype for test1 */
INST_DECL(vm,"E1411",REGISTER);   /*declare instrument variable  */

main()
{
  int fail;

  INST_STARTUP();                 /* initialize instrument       */
                                  /* operating system            */
  INST_OPEN(vm,"vxi,(24,25)");    /* open scanning meter using   */
                                  /* 1411 logical address 24, and*/
                                  /* scanner logical address 25  */
  if (!vm)
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  {
    printf("open vm failed, error number: %d\n",cscpi_open_error);
    exit(1);
  }
  fail = test1();                   /* run test using test1()      */
                                    /* function                    */
  if (fail)                         /* check to see if test passed */
  {                                 /* or failed                   */
    printf("TEST FAILED \n");
  }
  else
  {
    printf("TEST PASSED\n");
  }
  exit(0);
}
int test1(void)                     /* a simple test function      */
{
#define POINTS 10                   /* number of points            */

  float a[POINTS];                  /* define expected points      */
  float expected[POINTS]={1.0, 1.0, 1.0, 5.0, 5.0, 0.0, 1.0,2.5,
                          9.0, 0.0};
  int fail = 0;
  int i;
                                    /* query for results and put   */
                                    /* into array                  */
  INST_QUERY (vm, "MEAS:VOLT? 10, (@100:109)","%f",&a);

  for (i=0;i<POINTS;i++)
  {
    if (fabs(a[i]-expected[i]) > .01)
    {
      printf("test point %d failed. Expected %f, measured %f\n",
             i, expected[i], a[i]);
      fail=1;
    }
  }
  return fail;
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming with an External File

This programming example shows how you can use the C-SCPI `INST_EXTERN` macro command to specify an external declaration. This command works very similarly to the ANSI C `extern` command. See Chapter 5 later in this guide for more information on the `INST_EXTERN` command.

Equipment Needed

-- Embedded VXI Controller
-- HP VXI Mainframe
-- HP E1326B or HP E1411B Multimeter

Program Description

This program uses the C-SCPI `INST_EXTERN` and ANSI C `extern` commands. Program **example2.cs** uses the `INST_EXTERN` command to declare the instrument id, *vm*, of the HP E1326B or HP E1411B Multimeter as an external variable. Program **example2.cs** represents a library where routines can be added to create your own driver for the Multimeter. This program needs to be compiled separately from the main program file, **example2a.cs**. Program **example2a.cs** uses the ANSI C `extern` command to declare the function, `test1()`, as external to the program. Once declared as external to **example2a.cs**, then it can be called and executed.

Use the following steps to preprocess, compile, and link these programs:

- Run your C programs through the C-SCPI preprocessor:

```
cscpipp example2.cs > example2.c
cscpipp example2a.cs > example2a.c
```

- Compile each file:

```
gcc -c [-g] -mthreads example2.c
gcc -c [-g] -mthreads example2a.c
```

- Link the files:

```
gcc [-g] -mthreads -o example2 example2.o
     example2a.o cscpi_error.o -lcscpi -lsicl
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Listing

```
/*example2.cs*/
/*This programming example uses the INST_EXTERN C_SCPI command.  */
/*                                                               */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <cscpi.h>                  /*Needed for Preprocessor    */

INST_EXTERN(vm,"E1411",REGISTER);   /*declare external instrument*/
                                    /* variable                  */
test1()                             /* a simple test function    */
{
#define POINTS 10                   /* number of points          */

  float a[POINTS];                  /* define expected points    */
  float expected[POINTS]={1.0, 1.0, 1.0, 5.0, 5.0, 0.0, 1.0,2.5,
                          9.0, 0.0};
  int fail = 0;
  int i;
                                    /* query for results and put */
                                    /* in an array               */
  INST_QUERY (vm, "MEAS:VOLT? 10, (@100:109)","%f",a);

  for (i=0;i<POINTS;i++)
  {
    if (fabs(a[i]-expected[i]) > .01)
    {
      printf("test point %d failed. Expected %f, measured %f\n",
              i, expected[i], a[i]);
      fail=1;
    }
  }
  return fail;
}
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/*example2a.cs*/
/*This programming example uses the extern ANSI C command to     */
/*execute test1() listed on the previous page.                    */

#include <stdio.h>
#include <math.h>

#include <cscpi.h>                /*Needed for Preprocessor       */
                                  /* commands.                    */
INST_DECL(vm,"E1411",REGISTER);   /*declare instrument variable   */

extern int test1();               /* declare function from other */
                                  /* file                        */
main()
{
  int fail;

  INST_STARTUP();                 /* initialize instrument os    */
  INST_OPEN(vm,"vxi,(24,25)");    /* open scanning meter using   */
                                  /* 1411 logical address 24, and*/
                                  /* scanner logical address 25  */
  if (!vm)
  {
   printf("open vm failed, error number: %d\n", cscpi_open_error);
   exit(1);
  }
  fail = test1();                 /* run test using test1()      */
                                  /* external function           */
  if (fail)                       /* check to see if test passed */
  {                               /* or failed                   */
    printf("TEST FAILED \n");
  }
  else
  {
    printf("TEST PASSED\n");
  }
  exit(0);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming with a C-SCPI Parameter List

This programming example shows how you can use the C-SCPI `INST_PARAM` macro command to pass instrument identifications to functions. See Chapter 5 for more information on the `INST_PARAM` macro command.

Equipment Needed

-- Embedded VXI Controller
-- HP VXI Mainframe
-- HP E1326B or HP E1411B 5 1/2-Digit Multimeter

Program Description

There are two separate files listed on the following pages. The first file, **example3.cs**, represents the main program. It uses the functions listed in the second file. The second file, **example3a.cs**, represents a file of functions that can be compiled separately from the main program. This file uses the C-SCPI `INST_PARAM` macro command to pass the instrument declaration to the functions. More functions can be added to create your own drivers for the HP Multimeter.

Use the following steps to preprocess, compile and link these programs:

- Run your C programs through the C-SCPI preprocessor:

```
cscpipp example3.cs > example3.c
cscpipp example3a.cs > example3a.c
```

- Compile each file:

```
gcc -c [-g] -mthreads example3.c
gcc -c [-g] -mthreads example3a.c
```

- Link the program files and cscpi_error file:

```
gcc [-g] -mthreads -o example3 example3.o example3a.o
    cscpi_error.o -lcscpi -lsicl
```

See “Overview of C-SCPI” in Chapter 2 for more information on each of these steps.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Listing

```
/*example3.cs*/
/*This example shows how you can use a file of your own drivers */
/*for a specific instrument.*/

#include <stdio.h>
#include <stdlib.h>
#include <cscpi.h>

#define VM_ADDR “vxi,24”
#define FUNCTION “VOLT:AC”
#define RANGE 8

extern void E1411_func(INST id, char *func);
extern void E1411_volt_range (INST id, double range);
extern float E1411_read (INST id);

main()
{
  float answer;

  INST_DECL (vm, “E1411”, REGISTER);

  INST_STARTUP ();
  INST_OPEN (vm, VM_ADDR);

  E1411_func (vm, FUNCTION);
  E1411_volt_range (vm, RANGE);
  answer = E1411_read (vm);
  printf (“Answer :  %f\n”,answer);
}
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/*example3a.cs*/
/*This example is a file of drivers that can be preprocessed and */
/*used as a library of function calls from other C programs.     */

#include <cscpi.h>                    /*Needed for INST commands */


/*This function can be used to specify the function of the       */
/*multimeter.*/
void E1411_func (INST_PARAM (id, “E1411”, REGISTER), char *func)
{
  INST_SEND (id, “FUNC %S” , func);
}


/*This function can be used to specify the voltage range of      */
/*the multimeter.*/
void E1411_volt_range (INST_PARAM (id, “E1411”, REGISTER),
    double range)
{
  INST_SEND (id, “VOLT:RANGE %f “, range);
}


/*This function can be used to read data from the multimeter.    */
float E1411_read (INST_PARAM (id, “E1411”, REGISTER))
{
  float result;

  INST_QUERY (id, “READ?”, “%f”, &result);
  return (result);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Storing Block Data in a Separate File (The -f Option)

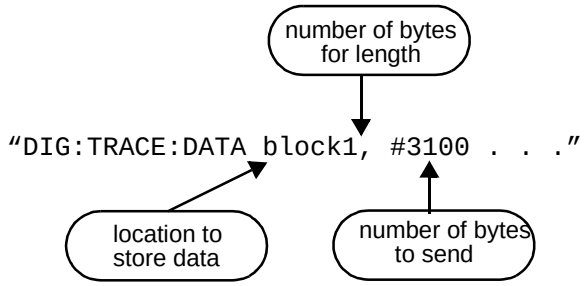
This programming example shows how you can use the `-f option` with C-SCPI to store block data in a separate file when the C-SCPI preprocessor runs. Using the `-f option` reduces the size and compilation time of your C program. See Chapter 2 for more information on the operation of the `-f option`.

Equipment Needed

-- Embedded VXI Controller
-- HP VXI Mainframe
-- HP E1330B 4-Channel Digital I/O

Program Description

This program contains a function called `load_traces()`. The following SCPI command sends block data to a defined memory block:



When this program is preprocessed with the C-SCPI `-f option`, the block data will be stored in a separate file (`example4.dat` as shown below):

```
cscpipp -f example4.dat example4.cs > example4.c
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In your program you must also open the file containing the block data and assign it to the `cscpi_datafile` FILE pointer for data retrieval. This is shown in the following example.

Program Listing

```
/*example4.cs*/
/*This program can be used with the -f option so that the block  */
/*data is stored in a separate file by the preprocessor.         */

#include <stdio.h>
#include <cscpi.h>

#define DIG_ADDR “vxi,144”


INST_DECL (dig, “E1330B”, REGISTER);  /*define instrument variable */
FILE *cscpi_datafile;                 /*must be used as pointer to */
                                      /*file opened for block data */
main ( )
{
  INST_STARTUP ();                    /*start operating system     */
  INST_OPEN (dig, DIG_ADDR);          /*initialize digital I/O card*/

  /*test to see if INST_OPEN worked, 0 returned in failed*/
  if (!dig)
  {
    printf (“Open FAILED. Error number: %d\n”, cscpi_open_error);
    exit (1);
  }
  /*open data file for block data*/
  cscpi_datafile = fopen (“example4.dat”, “rb”);

  /*test to see if fopen worked*/
  if (!cscpi_datafile)
  {
    printf (“Open example4.dat failed\n”);
    exit (1);
  }
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  /*call function with block data*/
  load_traces ();

  /*put your test here*/
  exit (0);
}


/*function to load traces*/
load_traces ( )
{
  /*This function generates code to send data to an instrument.*/
  /*When the C-SCPI preprocessor is run, the data is put in the*/
  /*file that cscpi_datafile is pointing to.*/

  /*set up memory block to send data*/
  INST_SEND (dig, “DIG:TRACE:DEF block1, 1000”);
  INST_SEND (dig, “DIG:TRACE:DEF block2, 1000”);

  /*send data to file, must be 100 bytes*/
  INST_SEND (dig, “DIG:TRACE:DATA block1, #3100”
     “12345678901234567890123456789012345678901234567890”
     “12345678901234567890123456789012345678901234567890”);


  INST_SEND (dig ,”DIG:TRACE:DATA block2, #3100”
      “abcdefghijklmnopqrstuvwxyz01234567890123456789012”
      “12345678901234567890123456789012345678901234567890”);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using C-SCPI in the Interactive Mode

Since the interactive functions allow you to enter the SCPI commands at run time, you can write your program to prompt the user for a SCPI command at run time. An example program using the `cscpi_exe` function is described in this section. This example is stored in the **/usr/hp75000/demos/cscpi** directory. See Chapter 5 for examples of the other two interactive functions.

Equipment Needed

-- Embedded VXI Controller
-- HP VXI Mainframe
-- HP E1326B or HP E1411B Digital Multimeter

Program Description

This program sets up and initializes the HP Multimeter. It prompts the user for an address and loops prompting for SCPI commands. The SCPI commands are parsed by the Controller and executed by the HP Multimeter. When a null string is entered, the loop is exited.

This program demonstrates how you can use the C-SCPI execute function in an interactive mode. See Chapter 5 for more details on the C-SCPI execute call.

Program Listing

```
/*example6.cs*/
/*This is a C-SCPI example of using the interactive mode. The      */
/*program is written to prompt the user for the SCPI command.      */
/*The command is then executed using the cscpi_exe function.       */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cscpi.h>

#define LENGTH 1000       /*maximum length of SCPI command & result*/

INST_DECL (vm, “E1411B”, REGISTER);    /*declaration for voltmeter  */

main()
{
  char command [LENGTH];              /*string variable for SCPI command*/
  char result [LENGTH];               /*string variable for result   */
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
  /*prompt user to enter logical address of multimeter*/
  puts ("Enter the logical address of the vm, for example,"
     " vxi,24: \n");
  gets (command);

  INST_STARTUP ();                    /*start operating system*/
  INST_OPEN (vm, command);            /*initialize multimeter*/

 /*if 0 returned, open failed and print message*/
  if (!vm)
  {
    fprintf (stderr, "vm at %s failed to open\n",command);
    exit (1);
  }

/*loop to enter SCPI commands*/

 for (;;)
  {
    printf ("Enter a SCPI command for the multimeter. Hit return "
      "to exit\n");
    while (!gets(command))        /*loop until get a nonzero size  */
      ;                           /*gets may terminate on interrupt*/
   if (!*command)                 /*caused by the command*/
      break;
   result[0] = 0;                 /*clear result string*/

   /*C-SCPI call to execute the SCPI command*/
   cscpi_exe (vm, command, strlen(command),result, sizeof(result));

   /*if you have a result, print it*/
   if (result[0])
       printf ("Result :  %s", result);
  }
  printf ("DONE\n");
  exit (0);
}
```

Note The char array returned for result includes a new line at the end of the array. Therefore, you do not have to include a new line when printing the results.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Triggering with the HP Embedded Computer

In order to use the external triggering on the HP Embedded Computer, you must route the external trigger lines to the TTL trigger lines. You must then edit your program to trigger from the TTL trigger lines instead of the external trigger lines. See the `ivxitrigroute` command in the SICL documentation for information on redirecting the trigger lines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4

Troubleshooting Compiled SCPI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting Compiled SCPI

This chapter provides a guide to troubleshooting errors that may occur when using Compiled SCPI software. Examples of the most common errors are described. This chapter includes the following:

-- Resolving Compiled SCPI Preprocessor Errors
-- Resolving Compile and Link Errors
-- Resolving Compiled SCPI Run-Time Errors
-- Using GNU Debugger
-- Trapping Errors with cscpi_error

Resolving Compiled SCPI Preprocessor Errors

When executing the preprocessor command, `cscpipp`, you may get preprocessor syntax or usage errors. These errors can occur from unfamiliarity with the C-SCPI command set or with just a simple typing error. The following illustrates executing the preprocessor command:

```
cscpipp example2.cs > example2.c
```

Syntax Error Example

Error Message

```
source2.cs", line 12: missing or extra "
```

Resolution

Look in your program on line 12 for a missing quote. Use Chapter 5 for command syntax. In this program snippet, the statement on line 12 is `INST_SEND`. The `INST_SEND` command is expecting a string constant containing SCPI commands. The string constant should be enclosed in quotes as displayed below:

```
INST_SEND(vm,"CONF:VOLT:DC %f",numb1);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Description



```
/* source2.cs: set voltmeter to measure DC volts, query for the */
/* results, and print the results.*/
#include <stdio.h>
#include <cscpi.h>
INST_DECL(vm,"E1411B",REGISTER);
main ()
{
  float numb1=2.0;
  float vm_dc;
  INST_STARTUP();
  INST_OPEN(vm,"vxi,24");
  INST_SEND(vm,"CONF:VOLT:DC %f,numb1);
  INST_QUERY(vm,"READ?","%f",&vm_dc);
  exit(0);
}
```

Where To Go For More Information

For C-SCPI preprocessor syntax errors, refer to the Chapter 5, “Compiled SCPI Command Reference.” Detailed descriptions of the C-SCPI commands and their usage are described there.

Usage Error Example

Error Message

```
"source2.cs", line 11:  undeclared identifier
"source2.cs", line 21:  undeclared identifier
```

Resolution

Look in your program on line 11 to determine the undeclared identifier. In this example, line 11 is as follows:

```
INST_OPEN(vm,"vxi,24");
```

The `INST_DECL` command does not precede the `INST_OPEN`, and therefore, `vm` has not been declared.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Description



```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <cscpi.h>

main ()
{
  float numb1=2.0;
  float vm_dc;
  INST_STARTUP();
  INST_OPEN(vm,"vxi,24");
  .
  .
  INST_DECL(vm,"E1411B",REGISTER);
  .
  .
  exit(0);
}
```

Where To Go For More Information

For C-SCPI preprocessor usage errors you should check the following:

-- Chapter 5, “Compiled SCPI Command Reference”

-- Chapter 2, “Using Compiled SCPI”.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Resolving Compile and Link Errors

Compile and link errors occur when executing the compiler and linker commands. Once you have resolved compile or link errors, you must re-run the C-SCPI preprocessor. The compile/link process is as follows:

compile `gcc -c -mthreads source2.c`

link
```
gcc -mthreads -o source2 source2.o
    -lcscpi -lsicl
```

Compile Error Examples

Error Message

```
source2.cs:5:parse error before 'vm'
source2.cs:5:warning:  data definition lacks type or storage class
source2.cs:In function main:
    .
    .
    .
```

Resolution

Determine if `vm` has been declared and defined as type `INST`. Check your program to determine if you have done the following:

1. included the `cscpi.h` header file to define type `INST`, and
2. used the `INST_DECL` command to declare `vm` as type `INST`.

For this particular error, the `cscpi.h` header file was not included.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Description



```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <stdlib.h>
INST_DECL(vm,"E1411B",REGISTER);
main ()
{
  float numb1=2.0;
  float vm_dc;
  INST_STARTUP();
  INST_OPEN(vm,"vxi,24");
  .
  exit(0);
}
```

Note

Once you have resolved a compile error, you **MUST** re-run the C-SCPI preprocessor. Otherwise, you will be compiling the same code you were before.

Error Message

```
source2.cs: In function main:
source2.cs:27: incompatible types in argument passing
```

Resolution

Look in your program on line 27 to determine the C-SCPI command in error. In this example, the C-SCPI command on line 27 is as follows:

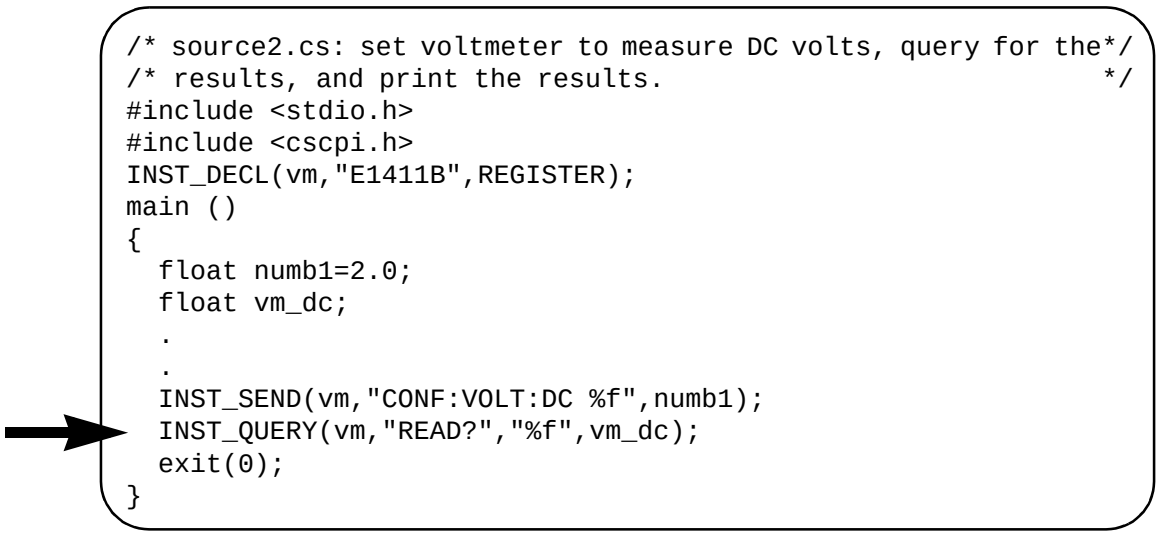
```
INST_QUERY(vm,"READ?","%f",vm_dc);
```

Using Chapter 5, “Compiled SCPI Command Reference” look up the `INST_QUERY` command and make sure you have set up the parameters correctly. The `INST_QUERY` command expects the parameter for the result of the query to be an address. Therefore, this parameter must be a pointer type. By placing the & in front of `vm_dc`, the results of the query will be placed in the address of the variable, `vm_dc`. The query statement should be as follows:

```
INST_QUERY(vm,"READ?","%f",&vm_dc);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Description



```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <cscpi.h>
INST_DECL(vm,"E1411B",REGISTER);
main ()
{
  float numb1=2.0;
  float vm_dc;
  .
  .
  INST_SEND(vm,"CONF:VOLT:DC %f",numb1);
  INST_QUERY(vm,"READ?","%f",vm_dc);
  exit(0);
}
```

Note

Once you have resolved a compile error, you **MUST** re-run the C-SCPI preprocessor. Otherwise, you will be compiling the same code you were before.

Where To Go For More Information

For this type of error you should check Chapter 5, “Compiled SCPI Command Reference” to review the syntax of the commands.

Error Message

```
source2.cs: In function main:
source2.cs:29: invalid type argument of ’unary’
```

Resolution

Look in your program on line 29 to determine the C-SCPI command in error. In this example, the C-SCPI command in error is as follows:

```
INST_QUERY(vm,"SYST:ERR?","%d,%s",err_num,err_msg);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using Chapter 5, “Compiled SCPI Command Reference” look up the `INST_QUERY` command and make sure you have setup the parameters correctly. The `INST_QUERY` command expects the parameter for the results of the query to be an address. Therefore, the parameters for both results should be a pointer type. The `SYST:ERR?` command differs from the `READ?` command in the previous example in that it returns two items: error number and error message. By placing the `&` in front of `err_num`, the results of the query will be placed in the address of the variable, `err_num`. The query statement should be as follows:

```
INST_QUERY(vm,"SYST:ERR?","%d,%s",&err_num,err_msg);
```

Because `err_msg` is declared as a character array (`char[]`), it does not require the `&` to be placed in front of it.

Note

Once you have resolved a compile error, you **MUST** re-run the C-SCPI preprocessor. Otherwise, you will be compiling the same code you were before.

Program Description

```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <cscpi.h>
INST_DECL(vm,"E1411B",REGISTER);
main ()
{
char err_msg[100];
int err_num;
.
.
INST_QUERY(vm,"SYST:ERR?","%d,%s",err_num,err_msg);
.
exit(0);
}
```

Where To Go For More Information

For this type of error you should check Chapter 5, “Compiled SCPI Command Reference” to review the syntax of the commands.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Link Error Examples

Error Message

```
Undefined Symbol(s):
iintron
iunmap
iintroff
imap
 .
 .
 .
```

Resolution

Determine if you have linked all the appropriate libraries your program needs. This particular error can be resolved by linking in the SICL library:

```
gcc -mthreads -o source2 source2.o -lcscpi -lsicl
```

Error Message

```
Undefined Symbol(s):
instr_misc
instr_query
instr_send
cscpi_open_error
 .
 .
 .
```

Resolution

Determine if you have linked all the appropriate libraries your program needs. This particular link error can be resolved by linking in the C-SCPI library:

```
gcc -mthreads -o source2 source2.o -lcscpi -lsicl
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Where To Go For More Information

For this type of error, you should refer to the following:

1. “Running Your First Compiled SCPI Program” in Chapter 1 or

2. Chapter 2, “Using Compiled SCPI” for information describing the compile and link process for C-SCPI.

Resolving Compiled SCPI Run-Time Errors

Run-time errors occur when executing the program's executable code. This section describes some run-time errors with descriptions of what to look for when trying to resolve them. For more detailed information on debugging run-time errors, refer to “Using GNU Debugger” beginning on page 79. Additionally, the instruments can generate errors. Refer to the “Trapping Errors with cscpi_error” beginning on page 81 of this chapter or the instrument manual for these specific errors.

Error Message

```
error: INST_OPEN before INST_STARTUP
open failed on vm
cscpi open error number: 1
```

Resolution

Look in your program to determine where this message is printed. Using Table 4-1, look up the `cscpi_open_error` number displayed to determine what could have caused this problem

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

.

Table 4-1. Run-time Errors

Error #	Most Likely Cause	Description of Cause
0	No error has occurred.	No error has occurred.
1	`INST_STARTUP` was **not** included in the program before `INST_OPEN`.	See following page for discussion.
2	A mismatch between the declaration and the open for the instrument(s) was detected by the instrument driver.	This occurs because the declaration made for the instrument (`INST_DECL`) did not match the instrument that was opened in the `INST_OPEN` command. This could also occur if one of the cards in a scanning multimeter is not supported.
3	System is out of memory.	Check your system's resources.
4	Format of the address encountered with a multiple card instrument was incorrect. Format: `INST_OPEN(vm,"vxi,(nn,nn)");`	This occurs when the software cannot understand an address of a multiple card instrument (scanning multimeter or Digital Functional Test System).
5	Invalid address was encountered (SICL `iopen("vxi,nn")` call failed.)	See **Program Description 2** of this section.
6	SICL is not setup properly or not running (SICL `iopen("vxi")` call failed).	Determine if SICL is running on your system. (See **Other Causes** of **Program Description 2** of this section.)
7 or 8	SICL has encountered a resource problem, or an Internal SICL error has occurred.	Contact your local Support organization.
9, 10, or 11	Internal SICL error has occurred.	Contact your local Support organization.
12 or 13	System encountered a resource problem.	Contact your local Support organization.
14	Instrument driver can not provide the required information.	This occurs when the instrument driver is not compatible with the version of C-SCPI that is installed on your system.

Note Once you have resolved a run-time error, you **MUST** re-run the C-SCPI preprocessor and re-compile and link your code. Otherwise, you will be using the same code you were before.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The `cscpi_open_error` variable is defined and declared in the `cscpi.h` include file. Therefore, you do not need to declare it in your program.

Program Description



```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <stdlib.h>
#include <cscpi.h>
INST_DECL(vm,"E1411B",REGISTER);
main ()
{
  float numb1=2.0;
  float vm_dc;
  INST_OPEN(vm,"vxi,24");
  if (vm==0)
  {
    printf("open failed on vm\n");
    printf("cscpi open error number: %d\n", cscpi_open_error);
    exit(1);
  }
  INST_SEND(vm,"CONF:VOLT:DC %f",numb1);
  INST_QUERY(vm,"READ?","%f",&vm_dc);
  exit(0);
}
```

This particular `cscpi_open_error` is error number 1. To resolve this error, you will need to add `INST_STARTUP` in the program before `INST_OPEN`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/* source2.cs: set voltmeter to measure DC volts, query for the*/
/* results, and print the results.                              */
#include <stdio.h>
#include <stdlib.h>
#include <cscpi.h>
INST_DECL(vm,"E1411B",REGISTER);
main (){
  float numb1=2.0;
  float vm_dc;
  INST_STARTUP();
  INST_OPEN(vm,"vxi,24");
  if (vm==0){
   .
   .
   .
}
```

Error Message

```
open failed on vm
cscpi open error number: 5
```

Resolution

This particular `cscpi_open_error` is error number 5. This error is reported because the instrument card does not exist at the address stated in the `INST_OPEN` command.

The program resolution is as follows:

```
.
main (){
.
  INST_STARTUP();
  INST_OPEN(vm,"vxi,24");
.
```

Note

Once you have resolved a run-time error, you **MUST** re-run the C-SCPI preprocessor and re-compile and link your code. Otherwise, you will be using the same code you were before.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Description



```
.
 main (){
   float numb1=2.0;
   float vm_dc;
   INST_STARTUP();
   INST_OPEN(vm,"VXI,32");
   if (vm==0){
     printf("open failed on vm\n");
     printf("cscpi open error number: %d\n",
 cscpi_open_error);
   exit(1);
 }
 .
```

Other Causes

Additionally, there are other reasons `INST_OPEN` may fail with an **error number 5** or **error number 6**. These are as follows:

-- SICL is not properly setup on the system where you are executing your program.

Where To Go For More Information

To resolve these errors you will need to do one or more of the following:

1. Use the error number from the `cscpi_open_error` variable to determine what type of problem you may have encountered.
2. Check Table 4-1 to look up possible causes of `cscpi_open_error`.
3. Run **/usr/sicl/bin/ivxisc** to determine if a resource manager problem occurred.
4. Refer to SICL documentation for more information on resolving SICL errors.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using GNU Debugger

This section provides a look at using the GNU debugger. GDB is a software tool that allows you to step through your program when a run-time error occurs. For more detailed information on how to use GDB, refer to Lynx Documentation.

When using GDB with your C-SCPI programs, you must compile and link your code using the debugger option, `-g`:

To Compile

```
gcc -c -g -mthreads source2.c
```

To Link

```
gcc -g -mthreads -o source2 source2.o
    -lcscpi -lsicl
```

To Compile/Link

```
gcc -g -mthreads -o source2 source2.c
    -lcscpi -lsicl
```

Preparing to Use GDB

Now, you can use `GDB` to debug your program. The following is a quick reference to some of the more useful `gdb` commands:

- "s" to single step INTO a function.
- "n" to single step OVER a function.
- "p" to print a variable or expression (p numb1).
- "p" to change value of a variable (p numb1=3.0).
- "c" to continue the program
- "b" to set a breakpoint at a line number (b 34).
- "list routine-name" to view the source code.
- "dir source-dir" to add a directory to the search path.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Executing GDB

To execute GDB, at the command prompt enter the following:

```
gdb source2
```

where `source2` is the name of your program.

You will enter the gdb program and will see the GDB prompt, **(gdb)**. At the prompt, you need to first send the command to ignore signal 32. Then you set a breakpoint (we have shown main as an example) and run. This will run the program to the breakpoint specified. You can then single step through your program with the s command. The following is an example of using GDB:

```
$gdb test
GDB is free software and you are welcome to distribute
copies of it under certain conditions; type “show
copying” to see the conditions. There is absolutely no
warranty for GDB; type “show warranty” for details.
GDB 4.15-96q1 (i386-unknown-lynxos2.5),
Copyright 1995 Free Software Foundation, Inc...
(gdb) b main
Breakpoint 1 at 0x267: file test.c, line 15.
(gdb)run
Starting program: /tmp/test
Kernel supports MTD ptrace requests.

Breakpoint 1, main () at test.c:15
15        id = iopen(“vxi”);
[New process 65 thread 37]
16        if (! id) {
(gdb)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trapping Errors with cscpi_error

This section provides a guide to using the `cscpi_error` routine to trap VXI instrument run-time errors. With C-SCPI, instrument syntax errors are reported when the C-SCPI preprocessor runs. Instrument run-time errors, however, are not reported which can result in incorrect program data.

You can write a `cscpi_error` routine and link it into your C program. This error routine would then be called every time an instrument run-time error is put into the instrument's error queue. If you do not use `cscpi_error`, C-SCPI provides a dummy routine that does nothing. Using `cscpi_error` allows you to write your own error routine to handle run-time errors.

Several advantages of using the `cscpi_error` routine include the following:

-- You assure data is correct by guaranteeing no errors are in the instrument's error queue.

-- You avoid continually checking the instrument's error queue for run-time errors.

Note `cscpi_error` is only called for `REGISTER` configuration types.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VXI Instrument Run-time Errors

A VXI instrument run-time error is an error that occurs at run time and is put into the instrument's error queue. Examples of run-time errors include out of range errors and triggering too fast errors.

The following program segment causes a VXI instrument run-time error.

```
/*This program segment sends a SCPI command to the HP E1411B*/
/*Multimeter that is out of range.*/

#include <stdio.h>
#include <cscpi.h>

INST_DECL (vm, "E1411B", REGISTER);

main()
{

  float answer;

  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");

  INST_SEND (vm, "FUNC:VOLT:AC");
  INST_SEND (vm, "VOLT:RANGE 400");
  INST_QUERY (vm, "READ?", "%f", &answer);


  printf ("answer: %f\n",answer);
}
```

The program will run and provide an answer. A run-time error, however, was generated and put into the instrument's error queue. You can add a `SYST:ERR?` command to check each instrument. The recommended method, however, is to use the `cscpi_error` routine to trap instrument run-time errors as they occur.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using cscpi_error

To use `cscpi_error` you must place a `cscpi_error` routine in your C program (or in a separate file and link it in). This routine is passed the SICL instrument id and the error number. The following `cscpi_error` routine can be found in the **/usr/hp75000/demos/cscpi** directory. You can use this error routine, edit this routine, or you can write your own.

```
/*cscpi_error.c*/
/*This routine provides the SCPI error routine for run-time errors.*/
/*Errors in the instrument's error queue are reported.*/

#include <cscpi.h>

void cscpi_error (INST sicl_inst, int error_number)
{
  char string[20]="SYST:ERR?";
  char result[255];

  cscpi_exe(sicl_inst,string,strlen(string),result,sizeof(result));

  printf ("ERROR:  %s",result);

  exit(1);
}
```

The `cscpi_error` routine provided prints the error message and exits the program. If you are using `cscpi_error` in a separate file, you must compile and link both files:

- Run your C program through the C-SCPI preprocessor:

  ```
  cscpipp example.cs > example.c
  ```

- Compile each file:

  ```
  gcc -c [-g] -mthreads example.c
  gcc -c [-g] -mthreads cscpi_error.c
  ```

- Link the files:

  ```
  gcc [-g] -mthreads -o example example.o cscpi_error.o
      -lcscpi -lsicl
  ```

When using `cscpi_error`, some restrictions apply. See the restrictions listed for the `INST_ONSRQ` command in Chapter 5.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notes:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5

Compiled SCPI Command Reference

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI Command Reference

This chapter contains a detailed description of the Compiled SCPI (Standard Commands for Programmable Instruments) command reference. This chapter contains the following sections:

Compiled SCPI Macro Commands

- `INST_CLEAR (`*id*`)` ..... Page 88
- `INST_CLOSE (`*id*`)` ..... Page 90
- `INST_DECL (`*id, driver, type*`)` ..... Page 92
- `INST_EXTERN (`*id, driver, type*`)` ..... Page 94
- `INST_ONSRQ (`*id, c_function*`)` ..... Page 96
- `INST_OPEN (`*id, dev_addr*`)` ..... Page 98
- `INST_PARAM (`*id, driver, type*`)` ..... Page 100
- `INST_QUERY (`*id, cmd_string, readfmt* [*c_expr…*],*c_addr*[,*c_addr…*]`)` Page 102
- `INST_READSTB (`*id, c_addr*`)` ..... Page 108
- `INST_SEND (`*id, cmd_string*[,*c_expr…*]`)` ..... Page 110
- `INST_STARTUP ()` ..... Page 116
- `INST_TRIGGER (`*id*`)` ..... Page 117

Compiled SCPI Functions

- `cscpi_error(`*sicl_inst, error_number*`)` ..... Page 119
- `cscpi_exe(`*id, cmd_string, cmd_length, result, result_length*`)` Page 120
- `cscpi_exe_fildes(`*id, in, out*`)` ..... Page 122
- `cscpi_exe_stream(`*id, fin, fout*`)` ..... Page 124
- `cscpi_get_overlap()` ..... Page 126
- `cscpi_overlap(`*mode*`)` ..... Page 127

Compiled SCPI Quick Reference ..... Page 128

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI Macro Commands

The C-SCPI macro commands are replaced by driver calls for `REGISTER` configurations and SICL commands for `MESSAGE` configurations when the C-SCPI preprocessor runs. Each C-SCPI macro command syntax is described in the following form:

C-SCPI Macro Command (*parameter1*, *parameter2*, [*parameter3*], *etc.*)

Commands are listed at the top of the page. Each command has a short description and the following information:

-- Command Syntax
-- Parameters
-- Comments
-- Example Program Segment

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_CLEAR

The instrument clear command sends an IEEE-488.2 device clear equivalent to the instrument selected by the *id* parameter.

Syntax

```
INST_CLEAR (id);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1410A", MESSAGE); /*assigns variable*/` `INST_CLEAR (vm);  /*clears E1410A*/`

Comments

- For `MESSAGE` configurations, C-SCPI uses the SICL `iclear(id)` function. See the SICL documentation for more information.
- With `REGISTER` configurations, you might use this command in a signal handling routine. You can set up the routine to be called in response to a signal that you generate, such as a Ctrl C. You might generate the Ctrl C if your program seems hung on an `INST_SEND` or `INST_QUERY` command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example sends a device clear to the HP E1410A Multimeter.

```
INST_DECL(vm, "E1410A", MESSAGE);
.
.
.
main()
{
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  INST_CLEAR(vm);
  .
  .
  .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_CLOSE

The instrument close command closes the I/O channel of communication with a device and releases memory used by the instrument driver.

Syntax

```
INST_CLOSE (id);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `INST_CLOSE (vm);  /*closes E1411B I/O channel*/`

Comments

- An `INST_OPEN` must be executed earlier in the program to open the I/O channel of communication and initialize the instrument driver.
- If you do not use an `INST_CLOSE` command, the I/O channel will automatically be closed when the program terminates. This is done by the underlying SICL I/O Library.
- Typical reasons for using the `INST_CLOSE` command include the following:
  - -- To open another I/O channel of communication for the same instrument.
  - -- To use an instrument in a different configuration. For example, you want to change a switch card configured as part of a scanning voltmeter to a switchbox configuration.
- For `MESSAGE` configurations, C-SCPI uses the SICL `iclose (id)` function. See the SICL documentation for more information.
- See also `INST_OPEN`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example opens an I/O channel of communication for a voltmeter and a switch. The I/O channels are then closed and an I/O channel is opened for a scanning voltmeter (with the switches configured as voltmeter cards). See the *HP E1326B/E1411B 5 1/2-Digit Multimeter User's Manual* for information on these configurations.

```
INST_DECL (vm, "E1411B", REGISTER);
INST_DECL (sw, "E1460A", REGISTER);
.
.
main()
{
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  INST_OPEN (sw, "vxi,(25,26)");
   .
   .
   .
  INST_CLOSE (vm);
  INST_CLOSE (sw);
   .
   .
  INST_OPEN (vm, "vxi,(24,25,26)");
   .
   .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_DECL

Instrument declare creates a variable declaration for the instrument data pointer. This variable can be declared a global variable in the mainline program, or it can be declared a local variable in a function. This command also designates the name of the instrument *driver* and the configuration *type* (`REGISTER` or `MESSAGE`).

Syntax

```
INST_DECL (id, driver, type);
```

Parameters

Parameter	Description
id	The user variable name for the instrument. This is where you declare the variable name for the instrument. This variable name will be used by other C-SCPI commands. For example: `INST_DECL (vm, "E1411B", REGISTER);` assigns vm to the E1411B.
driver	The parameter that defines the HP driver. This parameter is a quoted string, for example, "E1411B" is the driver name for the HP E1411B Multimeter. See the *HP 75000 Family of VXI Products* catalog for a list of available drivers (available from your nearest HP Sales and Service Office), or see Appendix A, “Online Documentation” for more information.
type	The parameter that defines the configuration type. The configuration type can be one of the following keywords: `MESSAGE` - for HP-IB or VXI message-based configurations `REGISTER` - for register-based configurations If you have a register-based card configured over HP-IB, the configuration *type* is `MESSAGE`.

Comments

- `INST_DECL`, `INST_EXTERN`, or `INST_PARAM` is required to declare the *id* parameter. This declaration must be done before any use of the *id* parameter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- The contents of the *driver* parameter are ignored for `MESSAGE` configurations. For those cards, use the instrument name (for example, E1410A) as the *driver* parameter. This will increase program readability.
- It is good practice to use `INST_DECL` at the beginning of your program to ensure you are not declaring an instrument in the middle of an executable statement. You can only use `INST_DECL` where you declare a normal C variable.
- See also `INST_EXTERN` and `INST_PARAM`.

Example

This example declares a global variable for the HP E1411B. Since the voltmeter is declared outside of a function, `vm` is available anywhere in the program file.

```
INST_DECL (vm, "E1411B", REGISTER);
.
.
main()
{
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  .
  .
}
```

This example declares a local variable for the HP E1411B. Since the voltmeter is declared in the function, `vm` is only available in the function `main`.

```
.
.
main()
{
  INST_DECL (vm, "E1411B", REGISTER);
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  .
  .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_EXTERN

Instrument external creates an external variable reference for an instrument data pointer. This command also specifies the instrument *driver* name and the configuration *type* (`REGISTER` or `MESSAGE`). This command is similar to instrument declare except that it declares a reference to an external variable.

Syntax

`INST_EXTERN` (*id, driver, type*);

Parameters

Parameter	Description
id	The user variable name for the instrument. This is where you declare the variable name for the instrument. This variable name will be used by other C-SCPI commands. For example: `INST_EXTERN (vm, "E1411B", REGISTER);` assigns `vm` to the HP E1411B.
driver	The parameter that defines the HP driver. This parameter is a quoted string, for example, "E1411B" is the driver name for the HP E1411B Multimeter. See the *HP 75000 Family of VXI Products* catalog for a list of available drivers (available from your nearest HP Sales and Service Office), or see Appendix A, “Online Documentation” for more information.
type	The parameter that defines the configuration type. The configuration type can be one of the following keywords: `MESSAGE` - for HP-IB or VXI message-based configurations `REGISTER` - for register-based configurations If you have a register-based card configured over HP-IB, the configuration *type* is `MESSAGE`.

Comments

- `INST_DECL`, `INST_EXTERN`, or `INST_PARAM` is required to declare the *id* parameter. This declaration must be done before any use of the *id* parameter.
- See also `INST_DECL` and `INST_PARAM`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example defines an external variable for the HP E1411B Multimeter. Two program segments are shown, one with the declaration and one with the `INST_EXTERN` to show vm declared in another file.

```
INST_DECL (vm, "E1411B", REGISTER);
.
extern void setup ();
 .
 .
main (){
 INST_STARTUP ();
 INST_OPEN (vm, "vxi,24");
  .
  .
 setup ();
  .
  .
}
```

```
INST_EXTERN (vm, "E1411B", REGISTER);
.
.
void setup()
{
  INST_SEND(vm, "*RST");
  .
  .
  .
}
```

See Chapter 3, “Programming with Compiled SCPI” for a more detailed example of this command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_ONSRQ

The instrument on service request command installs the function specified by the *c_function* parameter as a handler to be called when the instrument specified by *id* asserts a service request.

Syntax

```
INST_ONSRQ (id, c_function);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `INST_ONSRQ (vm, services;/*calls service when E1411B asserts service request*/`
c_function	The C function that is called when a service request is asserted from the instrument defined by the *id* parameter. When writing your C function, you must include the `INST_PARAM` command so that you can pass the instrument id. See the example shown on the next page.

Comments

- To disable the service request, set the *c_function* parameter to 0. It is good practice to disable service request when entering a service routine. This avoids any confusion that would result by calling the service routine while already executing this routine.
- If an SRQ function is called, it should generally not send commands to an instrument if the main program is using the same instrument. Also, some register-based instrument commands should not be executed from an SRQ function called from an interrupt routine. See the instrument's C-SCPI online documentation for information on when a register-based instrument causes an SRQ function to be called from an interrupt routine. See the SICL documentation for more information on message-based devices.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- The SICL `iintron` and `iintroff` functions can NOT be used for disabling and enabling the C-SCPI ONSRQ interrupts in `REGISTER` configurations. `REGISTER` configurations may call the *c_function* specified in the `INST_ONSRQ` macro command from within other `INST` functions. `MESSAGE` configurations, however, use the SICL interrupts and can be enabled or disabled using these SICL functions.
- For `MESSAGE` configurations, C-SCPI uses the SICL `ionsrq()` function. See the SICL documentation for more information.
- When writing your C function to be called with a service request, you must include the `INST_PARAM` command to pass the instrument id.
- If you use `INST_ONSRQ` while in the overlapped mode, you can come across other problems where your controller may be doing other tasks rather than finishing up the instrument command. See “Overlapped Mode” in Chapter 2 for additional information.
- The `INST_ONSRQ` command cannot be executed before an `INST_OPEN` command.
- See the `*SRE` command in the VXI instrument User's Manual.

Example

This example shows the call service if the HP E1411B Multimeter asserts a service request.

```
INST_DECL (vm, "E1411B", REGISTER);
.
void service (INST_PARAM (id, "E1411B", REGISTER))
{
 .
}
main ()
{
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  INST_ONSRQ (vm, service);
 .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_OPEN

The instrument open command opens the I/O channel of communication to the instrument specified by the *id* parameter. `REGISTER` configured instruments are initialized and set to their power-on state.

Syntax

```
INST_OPEN (id, dev_addr);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `INST_OPEN (vm, vxi,24);  /*opens I/O channel to E1411B*/`
dev_addr	The SICL address for the instrument. This address can be a quoted string or a pointer to a char array. With a pointer to a char array, you can assign card addresses at run time. If a VXI instrument is used, the "vxi,xx" quoted string should be used with xx being the logical address of the instrument. Multiple cards can be configured as a switchbox or scanning voltmeter. To specify multiple cards, enclose the card addresses in parentheses and separate with commas. For example, "vxi,(24,25,26)" can be used for a scanning voltmeter configuration. The voltmeter is at logical address 24, and the switch cards are at logical addresses 25 and 26. If an HP-IB instrument is used, the "hpibxx,xx,xx" quoted string should be used with the first xx being the logical unit, the second xx the primary address, and the last xx the secondary address. For example, "hpib7,23,1" can be used where 7,23,1 is the logical unit, primary address, and secondary address respectively.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comments

- This command sets `REGISTER` configured instruments to their power-on state. See the instrument's VXI user's manual for information on the instrument power-on state.
- `INST_OPEN` must be executed after the instrument declaration and before the system start-up (for example, `INST_DECL` and `INST_STARTUP`). This command must also be executed before any instrument command that uses *id*.
- If this command fails, the *id* parameter is set to 0. It is good practice to check the *id* parameter before doing anything else. For `REGISTER` configurations, you can also check the `cscpi_open_error` global variable for more information. (See Chapter 4, "Troubleshooting Compiled SCPI," for information on checking this variable). For `MESSAGE` configurations, see the SICL `igeterrno ()` and `igeterrstr ()`.
- For `MESSAGE` configurations, C-SCPI uses the SICL `iopen()` function. The command opens a connection to the instrument without performing an initialization. See the SICL documentation for more information.
- See also `INST_CLOSE`.

Example

This example initializes the HP E1411B and HP 3457A Multimeters and checks for errors.

```
INST_DECL(vm1, "E1411B", REGISTER);
INST_DECL (vm2, "3457A", MESSAGE);
main()
{
  INST_STARTUP();
  INST_OPEN (vm1, "vxi,24");
 if (vm1 == 0)
  {
    printf ("INST_OPEN failed,C-SCPI error:
      %d\n",cscpi_open_error);
    exit(1);
  }
  INST_OPEN (vm2, "hpib,22");
   .
   .
   .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_PARAM

The instrument parameter command defines an I/O channel to be passed to functions. This command is used when the instrument is declared out of scope and you need to pass the instrument declaration into a function.

Syntax

```
INST_PARAM (id, driver, type);
```

Parameters

Parameter	Description
id	The user variable name for the instrument. This is where you declare the variable for the instrument. This variable name will be used by other C-SCPI commands. For example: `INST_PARAM (vm, "E1411B", REGISTER);` assigns `vm` to the E1411B.
driver	The parameter that defines the HP driver. This parameter is a quoted string, for example, "E1411B" is the driver name for the HP E1411B Multimeter. See the *HP 75000 Family of VXI Products* catalog for a list of available drivers (available from your nearest HP Sales and Service Office), or see Appendix A, “Online Documentation” for more information.
type	The parameter that defines the configuration type. The configuration type can be one of the following keywords: `MESSAGE` - for HP-IB or VXI message-based configurations `REGISTER` - for register-based configurations If you have a register-based card configured over HP-IB, the configuration *type* is `MESSAGE`.

Comments

- `INST_DECL`, `INST_EXTERN`, or `INST_PARAM` is required to declare the *id* parameter. This declaration must be done before any use of the *id* parameter.
- See also `INST_DECL` and `INST_EXTERN`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example shows the `INST_PARAM` command used to pass the instrument declaration to the `setup` function.

```
setup (INST_PARAM (id, "E1411B", REGISTER))
{
  .
  .
  INST_SEND (id, "*RST");
  .
  .
}
main()
{
  INST_DECL (vm, "E1411B", REGISTER);
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  .
  .
  setup (vm);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_QUERY

The instrument query command sends the SCPI command(s) in the *cmd_string* parameter to the instrument defined by the *id* parameter. This command also stores the result(s) from the query command in the address(es) specified by the *c_addr* parameter.

Syntax

`INST_QUERY (`*id, cmd_string, readfmt* [, *c_expr*…], *c_addr* [, *c_addr*…]`);`

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);  /* assigns variable*/` `INST_QUERY (vm, "*TST?", "", &results);  /*sends to E1411B*/`
cmd_string	The string constant containing the SCPI command(s). See the instrument's VXI User's Manual for information on the SCPI commands, or see Appendix A, "Online Documentation" For example: `INST_QUERY (vm, "*TST?", "", &result);` where `*TST?` is the *cmd_string*. Any SCPI parameter in the string can be expressed with a format specifier. If a format specifier is used, each [*c_expr*] parameter that follows contains the corresponding C expression. This is similar to the format specifiers in the `C printf` function. The following format specifiers are available for `REGISTER` configurations: **%d %S %a** **%f %r** **%s %b** Each of these format specifiers is discussed on the following pages.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cmd_string Format Specifiers	**NOTE:** The format specifier must be the same type that the instrument expects. If, for example, the instrument expects a float number, you must use the `%f` format specifier. See the instrument user's manual for information on what the SCPI command expects. **NOTE:** For `MESSAGE` configurations, C-SCPI uses the SICL `iprintf` function. See the SICL documentation for additional information on the format specifiers for `MESSAGE` configurations.
	%d The **%d** format indicates that an int is used as the numeric expression for the [*c_expr*] parameter. For example, `int numb1;` `INST_QUERY (vm,"MEAS:RES? %d","%f",numb1,&result);` If a comma is present in the format specifier, the [*c_expr*] is a pointer to an array of integers instead of a numeric expression. The comma operator is immediately followed by a number indicating the size of the array. The comma can be useful when you have a list of channels to query. For example, `int list[5] ={101,102,103,104,105};` `INST_QUERY(sw,"CLOSE? (@%,5d)","%,5d",list,&result);` **NOTE:** C-SCPI requires that the comma operator's corresponding array contain the complete channel list. In the C-SCPI command above, for example, you cannot add another comma operator to specify more channels to query.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

<table>
<tr><td>cmd_string Format Specifiers (continued)</td><td>%f</td><td>The %f format indicates that a float is used as the numeric expression for the [c_expr] parameter. For example,

<code>float numb1;
INST_QUERY (vm,"MEAS:RES? %f","%f",numb1,&result);</code>

If a comma is present in the format specifier, the [c_expr] is a pointer to an array of floats instead of a numeric expression. The comma operator is immediately followed by a number indicating the size of the array. The comma can be useful when you have a list of channels to query. For example,

<code>float list[5] ={101,102,103,104,105};
INST_QUERY(sw,"CLOSE? (@%,5f)","%,5d",list,&result);</code>

NOTE: C-SCPI requires that the comma operator's corresponding array contain the complete channel list. In the C-SCPI command above, for example, you cannot add another comma operator to specify more channels to query.</td></tr>
<tr><td></td><td>%s</td><td>The %s format indicates that a string expression without quotations is used in [c_expr]. For example,

<code>char count[5] = "MAX";
.
INST_QUERY (vm,"SAMP:COUNT? %s","%d",count,&result);</code></td></tr>
<tr><td></td><td>%S</td><td>The %S format indicates a string expression with quotations is used for [c_expr]. See <code>INST_SEND</code> for example.</td></tr>
<tr><td></td><td>%r</td><td>The %r format indicates an expression is used for [c_expr]. See <code>INST_SEND</code> for example.</td></tr>
<tr><td></td><td>%<<i>width</i>>[<i>size</i>]b</td><td>The %b format indicates that an array is used for [<i>c_expr</i>]. <<i>width</i>> is either a number or an asterisk (*). The number indicates the number of elements to be sent, and the * indicates that the number is taken from the next parameter (for example, <code>%1024b</code> or <code>%*b</code>). See <code>INST_SEND</code> for an example. [<i>size</i>] determines the size of each element in the array. [<i>size</i>] can be one of the following:

%b pointer to an array of char (8 bits)
%lb pointer to an array of long int (32 bit words)
%wb pointer to an array of short (16 bit words)
%zb pointer to an array of float (32 bit floating point numbers)
%Zb pointer to an array of double (64 bit floating point numbers)</td></tr>
</table>

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cmd_string Format Specifiers (*continued*)	**%<*width*>** [***size***]**a** The **%a** format indicates that a binary FILE* is used for [*c_expr*]. <*width*> is either a number or an asterisk (*). The number indicates the number of elements to be sent, and the * indicates that the number is taken from the next parameter (for example, %1024a or %*a). See INST_SEND for an example. [*size*] determines the size of each element in the array. [*size*] can be one of the following: **%a** binary file containing an array of char (8 bits) **%la** binary file containing an array of long int (32 bit words) **%wa** binary file containing an array of short (16 bit words) **%za** binary file containing an array of float (32 bit floating point) **%Za** binary file containing an array of double (64 bit floating point)
readfmt	The format of the query result when using a MESSAGE configuration. A format specifier is used in this location. See the instrument's VXI user's manual for information on query response types. See iscanf in the SICL manual for a description of available format specifiers. `INST_QUERY (vm, "*IDN?", "%s", addrloc);` indicates that the result of the *IDN? command is a string (because of the %s). The contents of this parameter are currently ignored for REGISTER configurations; however, it is good practice to include a format specifier in case you change to a MESSAGE configuration.
[***c_expr***]	The expression that is used if a format specifier appears in the *cmd_string* parameter. Any valid C expression can be used. The preprocessor does not check to make sure it is the same type as the format specifier.
c_addr	The address where the results of the instrument query is stored. Any valid C expression which evaluates to an address can be used. The preprocessor does not check it. The address location should be of the same type as the response. If a string is returned, only non-quoted strings are returned. See Appendix A, "Online Documentation" for details on finding query command response types.
[***c_addr***]	The address if more than one result is returned from the query. For example, if you have "SYST:ERROR?" as the SCPI command, two items are returned, the error number and the error message: `INST_QUERY (vm, "SYST:ERR?","",&err_num,err_msg);`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comments

- Multiple SCPI commands can be combined in the *cmd_string* parameter. As with SCPI, these commands are separated by semicolons (;). For example, `*RST;MEAS:VOLT:AC?` is a common command (`*RST`) linked with a SCPI command (`MEAS:VOLT:AC?`) separated by a semicolon.
- The *cmd_string* parameter must be a quoted string, and the SCPI command can not be a variable. If you want to use a string variable, use the `cscpi_exe` function call.Since HP VXI instruments and the Controllers use different types of microprocessors, the byte ordering is critical for `%b` and `%a` blocks. SICL takes care of the `MESSAGE` configurations. For `REGISTER` configurations, however, you must know what data size (byte, long, etc.) the instrument expects. See the C-SCPI online documentation for the specific instrument and use the format specifiers to specify the size of the data type.
- `REGISTER` configurations currently ignore the contents of *readfmt*. This parameter is used by `MESSAGE` configurations to specify the format of the query result. However, you can still use this parameter in `REGISTER` configurations for program readability.
- When you send a SCPI command to a `MESSAGE` configuration, you may have to include an end of line terminator (`\n` or `;`) if the instrument expects one.
- Online documentation is provided. The documentation contains a SCPI quick reference, commands not supported, commands changed, the SCPI command query response types, a list of overlapping commands, and ONSRQ restrictions. Documentation is supplied for each supported card. See Appendix A, “Online Documentation” for more information.
- For `MESSAGE` configurations, C-SCPI uses the SICL `ipromptf()` function. See the SICL documentation for more information.
- See also `INST_SEND`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example queries for a card description and stores the result. The result of the card description query is stored in `result`.

```
INST_DECL (vm, "E1411B", REGISTER);
main(){
  int cardnum=1;
  char result[255];
  .
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  INST_QUERY (vm, "SYSTEM:CDES? %d", "%s" ,cardnum,
     &result);
  printf ("the vm card description: %s\n", result);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_READSTB

The instrument read status byte command places the results of a serial poll from the instrument specified by the *id* parameter in the address specified by the *c_addr* parameter.

Syntax

```
INST_READSTB (id, c_addr);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `INST_READSTB (vm, &stb);  /*polls E1411B*/`
c_addr	The address where the results of the serial poll are stored. Any valid C expression which evaluates to an unsigned character pointer can be used. The preprocessor does not check to make sure that the *c_addr* is an unsigned character pointer.

Comments

- The *c_addr* parameter must evaluate to an unsigned character pointer.
- For `MESSAGE` configurations, C-SCPI uses the SICL `ireadstb()` function. See the SICL documentation for more information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example polls the HP E1411B and stores the results in the `stb` address location.

```
INST_DECL (vm, "E1411B", REGISTER);
main()
{
  unsigned char   stb;
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  .
  .
  INST_READSTB (vm, &stb);
 .
 .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_SEND

The instrument send command sends the SCPI information in the *cmd_string* parameter to the instrument defined by the *id* parameter.

Syntax `INST_SEND (`*id*, *cmd_string* [, *c_expr*...]`);`

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);  /* assigns variable*/` `INST_SEND (vm, "*RST");  /*sends to E1411B*/`
cmd_string	The string constant containing the SCPI command(s). See the instrument's VXI user's manual for information on the SCPI commands, or see Appendix A, "Online Documentation," for more information. For example: `INST_SEND (vm, "*RST");` where `*RST` is the *cmd_string*. Any SCPI parameter in the string can be expressed with a format specifier. If a format specifier is used, each [*c_expr*] parameter that follows contains the corresponding C expression. This is similar to the format specifiers in the `C printf` function. The following format specifiers are available for `REGISTER` configurations: **`%d`** **`%S`** **`%a`** **`%f`** **`%r`** **`%s`** **`%b`** Each of these format specifiers is discussed on the following pages.
cmd_string Format Specifiers	**NOTE:** The format specifier must be the same type that the instrument expects. If, for example, the instrument expects a float number, you must use the `%f` format specifier. See the instrument's user's manual for information on what the SCPI command expects. **NOTE:** For `MESSAGE` configurations, C-SCPI uses the SICL `iprintf` function. See the SICL documentation for additional information on the format specifiers for `MESSAGE` configurations

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cmd_string Format Specifiers (*continued*)	**%d**	The **%d** format indicates that an int is used for the numeric expression for the [*c_expr*] parameter. For example, `int numb1 = 5;` `INST_SEND(vm, "CONF:VOLT:DC %d", numb1);` If a comma is present in the format specifier, the [*c_expr*] is a pointer to an array of integers instead of a numeric expression. The comma operator is immediately followed by a number indicating the size of the array. The comma can be useful when you have a list of channels to close. For example, `int list[5] ={101,102,103,104,105};` `INST_SEND (sw,"CLOSE (@%,5d)", list);` **NOTE:** C-SCPI requires that the comma operator's corresponding array contain the complete channel list. In the C-SCPI command above, for example you cannot add another comma operator to specify more channels to be closed.
	%f	The **%f** format indicates that a float is used for the numeric expression for the [*c_expr*] parameter. For example, `float numb1 = 5.2;` `INST_SEND(vm, "CONF:VOLT:DC %f", numb1);` If a comma is present in the format specifier, the [*c_expr*] is a pointer to an array of floats instead of a numeric expression. The comma operator is immediately followed by a number indicating the size of the array. The comma can be useful to define a waveform for the HP E1340A Arbitrary Function Generator. For example, `float list[5] ={.5, 1.0, .5, 0,-0.5};` `INST_SEND (arb,"LIST:VOLT %,5f", list);` **NOTE:** C-SCPI requires that the comma operator's corresponding array contain the complete list. In the C-SCPI command above, for example, you cannot add another comma operator to specify more points in the waveform.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cmd_string Format Specifiers (*continued*)	**%s**	The **%s** format indicates that a string expression without quotations is used in [*c_expr*]. For example, `char count [5] = "MAX";` `.` `.` `INST_SEND(vm, "SAMP: COUNT %s", count) ;`
	%S	The **%S** format indicates a string expression with quotations is used for [*c_expr*]. The preprocessor will add the quotes. For example, `char function[22] = "VOLT:AC";` `.` `.` `INST_SEND(vm, "FUNC %S", function);` where the preprocessor puts `VOLT:AC` in the `%S` location with quotes around it, resulting in `FUNC "VOLT:AC"`.
	%r	The **%r** format indicates an expression is used for [*c_expr*]. For example, `char cond[] ="Q4 or Q5 or Q6";` `INST_SEND(d20, "DIG:TIM:COND:DEF %r", cond););` where `cond` is enclosed in parentheses, `(Q4 or Q5 or Q6)`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cmd_string Format Specifiers (*continued*)	**%<*width*>[*size*]b**	The **%b** format indicates that an array is used for [*c_expr*]. <*width*> is either a number or an asterisk (*). The number indicates the number of elements to be sent, and the * indicates that the number is taken from the next parameter (for example, %1024b or %*b). For example, `char data[1024];` `INST_SEND (dig,"SOUR:DIG:TRAC:DATA block,%1024b", data);` where, 1024 bytes of the array called data is used. You can also use an int or float array since C-SCPI casts it to a char array. [*size*] determines the size of each element in the array. [size] can be one of the following: **%b** pointer to an array of char (8 bits) **%lb** pointer to an array of long int (32 bit words) **%wb** pointer to an array of short (16 bit words) **%zb** pointer to an array of float (32 bit floating point numbers) **%Zb** pointer to an array of double (64 bit floating point numbers)
	%<*width*>[*size*]a	The **%a** format indicates that a binary FILE* is used for [*c_expr*]. <*width*> is either a number or an asterisk (*). The number indicates the number of elements to be sent, and the * indicates that the number is taken from the next parameter (for example, %1024a or %*a). For example, `int numb = 1024;` `INST_SEND (dig,"SOUR:DIG:TRAC:DATA block,%*a", numb,datafile);` where, the first 1024 bytes of the file are sent to the dig. [*size*] determines the size of each element in the array. [*size*] can be one of the following: **%a** binary file containing an array of char (8 bits) **%la** binary file containing an array of int (32 bit words) **%wa** binary file containing an array of short (16 bit words) **%za** binary file containing an array of float (32 bit floating point) **%Za** binary file containing an array of double (64 bit floating point)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

[***c_expr***]	The expression that is used if a format specifier appears in the *cmd_string* parameter. Any valid C expression can be used. The preprocessor does not check to make sure it is the same type as the format specifier.

Comments

- Multiple SCPI commands can be combined in the *cmd_string* parameter. As with SCPI, these commands are separated by semicolons (;). For example, `*RST;SENS:FUNC:VOLT:AC` is a common command (`*RST`) linked with a SCPI command (`SENS:FUNC:VOLT:AC`) separated by a semicolon.
- The *cmd_string* parameter must be a quoted string. The SCPI command cannot be a variable. If you want to use a string variable, use the `cscpi_exe` function call.
- Since HP VXI instruments and the Controllers use different types of microprocessors, the byte ordering is critical in the `%b` and `%a` blocks. SICL takes care of the `MESSAGE` configurations. For `REGISTER` configurations, however, you must know what data size (8, 16, 32, etc.) the instrument expects. See the C-SCPI online Documentation for the specific instrument and use the format specifiers to specify the size of the data type.
- For `MESSAGE` configurations, C-SCPI uses the SICL `iprintf` function. See the SICL documentation for more information.
- When you send a SCPI command to a `MESSAGE` configuration, you may have to include an end of line terminator (`\n` or `;`) if the instrument expects one.
- Online documentation is provided. The documentation contains a SCPI quick reference, commands not supported, commands changed, the SCPI command query response types, a list of overlapping commands, and ONSRQ restrictions. Online documentation is supplied for each supported card. See Appendix A, “Online Documentation” for additional information.
- See also `INST_QUERY`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example sends the select measurement function to the HP E1411B Multimeter. The %S format specifier indicates that `type` is declared as a quoted string.

```
INST_DECL (vm, "E1411B", REGISTER);
main()
{
  char type[10]="FRES";
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  INST_SEND (vm, "SENSE:FUNCTION %S", type);
  .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_STARTUP

The instrument start-up command starts the register-based operating system.

Syntax

```
INST_STARTUP ();
```

Comments

- This command must be executed before any other `REGISTER` configured commands.

Example

This example shows a common place to use the instrument start-up command.

```
INST_DECL(vm, "E1411B", REGISTER);
main()
{
  INST_STARTUP();
  .
  .
  .
  INST_OPEN (vm, "vxi,24");
  INST_CLEAR(vm);
  .
  .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INST_TRIGGER

The instrument trigger command sends an IEEE-488.2 equivalent group execute trigger to the instrument specified by the *id* parameter.

Syntax

```
INST_TRIGGER (id);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `INST_TRIGGER (vm);  /*Triggers E1411B*/`

Comments

- For `MESSAGE` configurations, C-SCPI uses the SICL `itrigger(id)` function. See the SICL for more information.

Example

This example shows the instrument trigger command being used.

```
INST_DECL(vm, "E1411B", REGISTER);
main()
{
  INST_STARTUP();
  INST_OPEN (vm, "vxi,24");
  .
  .
  INST_TRIGGER(vm);
  .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI Functions

The Compiled SCPI functions are provided as additional set of functionality. Each function is described in the following form:

C-SCPI Function (*parameter1*, *parameter2*, [*parameter3*], *etc.*)

The C-SCPI function is listed at the top of the page. Each function has a short description and the following information:

-- Function Syntax
-- Parameters
-- Comments
-- Example Program Segment

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_error

The C-SCPI error function provides error trapping for instrument run-time errors. Every time a run-time error is put into an instrument's error queue, a dummy `cscpi_error` function is called. However, if you write your own `cscpi_error` function and link it into your program, your function will be called instead. We have provided an example error routine you can use in your programs.

Syntax

`cscpi_error (`*sicl_inst, error_number*`)`

Example

The example `cscpi_error` routine provided is called when a run-time error is put into an instrument's error queue. This routine prints the instrument's error number and message. If you use this error routine, you must compile and link it into your main program.

```
/*cscpi_error.c*/
/*This routine provides the SCPI error routine for run time errors.*/
/*Errors in the instrument's error queue are reported.*/

#include <cscpi.h>
void cscpi_error (INST sicl_inst, int error_number)
{
  char string[20]="SYST:ERR?";
  char result[255];

  cscpi_exe(sicl_inst,string,strlen(string),result,sizeof(result));

  printf ("ERROR:  %s",result);
  exit(1);
}
```

See Chapter 4, "Troubleshooting Compiled SCPI," for more information on the `cscpi_error` routine.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_exe

The Compiled SCPI execute function allows you to use an interactive mode of operation with `REGISTER` configurations only. With this function the SCPI commands are parsed and executed at run time.

Syntax

`cscpi_exe (`*id, cmd_string, cmd_length, result, result_length*`);`

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `cscpi_exe (vm, command, strlen(command),` `result,sizeof(result)); /*sends to E1411B*/`
cmd_string	The string variable containing the SCPI commands. See the instrument's VXI User's Manual for information on the SCPI commands, or see the instrument's C-SCPI online documentation.
cmd_length	The length of the *cmd_string* parameter. If you are using this interactively, you can use the C string length function (`strlen`) to get this length.
result	The address location where the results of an instrument query are to be stored. The address location must be a pointer of type char [].
result_length	The maximum length of the returned string. If you are using this call interactively, you can use the C size of function (`sizeof()`) to get this length.

Comments

- This function is only interactive if you write it into your program to prompt the user for SCPI commands. See "Interactive Functions" in Chapter 2 for a more detailed example of `cscpi_exe`.
- The char array returned for *result* includes a new line at the end of the array. Therefore, you do not have to include a new line if you print the results.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example shows `cscpi_exe` being used in a loop prompting for input.

```
INST_DECL (vm, "E1411B", REGISTER);
  .
  .
main ()
{
  char result [255];
  char command [255];
      .
      .
      .
  INST_STARTUP ();
  INST_OPEN (vm, "vxi,24");
  for (;;)
  {
    printf("Enter Command\n");
    gets(command);
    cscpi_exe (vm, command, strlen(command), result,
       sizeof(result));
    if (result[0]!=0)
      printf ("The result of the %s command is : %s",
        command, result);
          .
          .
          .
      }
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_exe_fildes

The C-SCPI execute file descriptor function allows you to specify a file descriptor as an input file or pipe that contains SCPI commands to be executed. The results of the executed SCPI commands are put into the file descriptor specified for output. This function only works with `REGISTER` configurations. With this function the SCPI commands are parsed and executed at run time.

Syntax

```
cscpi_exe_fildes (id, in, out);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `cscpi_exe_fildes (vm,0, 1);  /*sends to E1411B*/`
in	An integer as the file descriptor to determine the input file. This file needs to contain SCPI commands to be executed. The SCPI commands must be for the `REGISTER` configured instrument specified in the *id* parameter.
out	An integer as the file descriptor to determine the output file. The results from the executed SCPI commands are stored in this file.

Comments

- Since `stdin` (0), `stdout` (1), and `stderr` (2) can be used, you can do terminal I/O without having to open extra files.
- ONLY SCPI commands for the instrument specified in the *id* parameter can be used in the input file.
- The input and output may be piped or redirected to other controlling processes. Connecting `stdin` (0) and `stdout` (1) to pipes allows you to have a separate process that is independent of SCPI or C-SCPI. See the Lynx manuals for more information on piping or redirecting to other processes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

This example uses the `stdin` (0) and `stdout` (1) file descriptors with `cscpi_exe_fildes`. The user can enter the SCPI commands from the terminal keyboard and the results are displayed on the terminal screen. The program continues until the user enters a <Ctrl D> which indicates the end of the file.

```
INST_DECL (vm, "E1411B", REGISTER);
  .
  .
main ()
{
  .
  .
  INST_STARTUP ();
  INST_OPEN (vm, "vxi,24");

  printf("Enter SCPI Commands for E1411B. Enter Ctrl D to quit\n");
  cscpi_exe_fildes (vm, 0,1);
  printf ("Done\n");
       .
       .
       .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_exe_stream

The C-SCPI execute stream function allows you to specify a file or pipe that contains SCPI commands to be used as input. The commands are executed and the results are stored in the file or pipe specified for output. This function only works with `REGISTER` configurations. With this function the SCPI commands are parsed and executed at run time.

Syntax

```
cscpi_exe_stream (id, fin, fout);
```

Parameters

Parameters	Description
id	The user variable name for the instrument. This is the variable name that you assigned to the instrument in the `INST_DECL` or `INST_EXTERN` command. Once the variable name is assigned, you can use that variable to send information to the instrument. For example, `INST_DECL (vm, "E1411B", REGISTER);/* assigns variable*/` `cscpi_exe_stream (vm, stdin, stdout);/*sends to E1411B*/`
fin	A FILE* that contains SCPI commands to be executed. The stream must contain ONLY SCPI commands for the `REGISTER` configured instrument specified in the *id* parameter. `stdin` can be used to get input from the terminal keyboard.
fout	A FILE* where the results of the executed SCPI commands will be stored. `stdout` or `stderr` can be used to display the results on the terminal display.

Comments

- Since `stdin`, `stdout`, and `stderr` can be used, you can do terminal I/O without having to open extra files.
- ONLY SCPI commands for the instrument specified in the *id* parameter can be used in the input file. The input and output may be piped or redirected to another controlling process. Connecting `stdin` and `stdout` to pipes allows you to have a separate process that is independent of SCPI or C-SCPI. See the Lynx manuals for more information on piping or redirecting to other processes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- If *fout* is a pipe, the c function `setbuf(fout, NULL);` must be used to disable buffering. If this function is not used, you will not get an output until the stream buffer is full. See the Lynx manuals for additional information.

Example

This example uses the `stdin` and `stdout` files with the `cscpi_exe_stream` function. The user can enter the SCPI commands from the terminal keyboard and see the results on the terminal screen. The program will continue to run until the user enters a <Ctrl D> which indicates the end of a file.

```
INST_DECL (vm, "E1411B", REGISTER);
  .
  .
  .
main ()
{
   .
   .
   .
  INST_STARTUP ();
  INST_OPEN (vm, "vxi,24");

  printf("Enter SCPI Commands for E1411B. Enter Ctrl D to quit\n");
  cscpi_exe_stream (vm, stdin, stdout);

   .
   .
   .
 printf ("Done\n");
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_get_overlap

The Compiled SCPI get overlap function returns an integer value that tells if the overlapped mode is ON (1) or OFF (0).

Syntax

```
cscpi_get_overlap ( ) ;
```

Example

This example stores the current status of the overlapped mode, turns overlapped ON, performs a function, and then returns overlapped mode to the status it had before.

```
main ()
{
  int old_mode;
  int new_mode =1;

  old_mode = cscpi_get_overlap ();
  cscpi_overlap (new_mode);
     .
     .
     .
  cscpi_overlap (old_mode);
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cscpi_overlap

The compiled SCPI overlap function turns the overlapped mode of operation ON or OFF. Turning this mode ON allows overlapping commands to be executed in parallel with other commands.

Syntax

```
cscpi_overlap (mode);
```

Parameters

Parameters	Description
mode	Determines if overlapped is ON or OFF. *mode* is an integer. Any non-zero value turns overlapped ON, and 0 turns it OFF.

Comments

- Before using the overlapped mode, see “Overlapped Mode” in Chapter 2 for information on the possible side effects.
- Overlapped mode is set to 0 (OFF) by the C-SCPI `INST_STARTUP ()` command.

Example

This example shows how to turn the C-SCPI overlapped mode ON.

```
main ()
{
  int mode = 1;
    .
    .
    .
  cscpi_overlap (mode);
     .
     .
     .
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI Quick Reference

Command Syntax	Command Description
`INST_CLEAR(`*id*`)`	Sends an IEEE-488.2 equivalent device clear to the instrument specified by the *id* parameter.
`INST_CLOSE(`*id*`)`	Closes the I/O channel of communication with a device and releases memory used by the instrument driver.
`INST_DECL(`*id, driver, type*`)`	Creates a variable declaration for the instrument specified by the *driver* parameter.
`INST_EXTERN(`*id, driver, type*`)`	Creates an external variable for the instrument specified by the *driver* parameter.
`INST_ONSRQ(`*id, c_function*`)`	Installs the function specified by the *c_function* parameter as a handler to be called when the instrument specified by *id* asserts a service request.
`INST_OPEN(`*id, dev_addr*`)`	Opens the I/O channel of communication with a device to the instrument specified by the *id* parameter.
`INST_PARAM(`*id, driver, type*`)`	Defines an I/O channel that will be passed to functions.
`INST_QUERY(`*id, cmd_string, readfmt* [,*c_expr…*], *c_addr*[,*c_addr…*]`)`	Executes the SCPI command in the *cmd_string* parameter addressed to the instrument specified by the *id* parameter. The query results are stored in the *c_addr* address.
`INST_READSTB(`*id, c_addr*`)`	Places the results of a serial poll from the instrument specified by the *id* parameter in the address location specified by the *c_addr* parameter.
`INST_SEND(`*id, cmd_string,* [,*c_expr…*]`)`	Executes the SCPI command in the *cmd_string* parameter addressed to the instrument specified by the *id* parameter.
`INST_STARTUP()`	Starts the register-based operating system.
`INST_TRIGGER(`*id*`)`	Sends an IEEE-488.2 equivalent group execute trigger to the register-based instrument specified by the *id* parameter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C-SCPI Calls	C-SCPI Call Description
`cscpi_error (`*sicl_inst, error_number*`)`	Provides error trapping for instrument run-time errors.
`cscpi_exe (`*id, cmd_string, cmd_length, result, result_length*`)`	Executes the SCPI *cmd_string* parameter at run time. The *cmd_string* parameter can be entered at run time.
`cscpi_exe_fildes (`*id, in, out*`)`	Executes SCPI commands using file descriptors as input and output to the command.
`cscpi_exe_stream (`*id, fin, fout*`)`	Executes SCPI commands using streams as input and output to the command.
`cscpi_get_overlap ()`	Returns an integer value if overlapped mode is ON (1) or OFF (0).
`cscpi_overlap (`*mode*`)`	Turns overlapped mode ON (1) or OFF (0).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notes:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A

Online Documentation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Online Documentation

One of the learning products C-SCPI provides is online documentation. We provide this online documentation in a form similar to UNIX® manual pages. By using the `man` command, you can get information on C-SCPI commands. This appendix describes the C-SCPI documentation and how to use it.

How To Use Manual Pages

To use manual pages you type the `man` command followed by a C-SCPI command or an HP register-based instrument model number:

```
man name
```

where name is either a C-SCPI Macro command, C-SCPI function call, or an HP register-based instrument model number. The following are examples of valid manual page commands:

```
man cscpipp

man INST_DECL

man cscpi_exe

man cscpi_drivers

man E1411B
```

What is a Manual Page?

A manual page is online documentation that is similar to the manual pages provided in the UNIX environment. Manual pages are provided for the following:

- C-SCPI Macro Commands
- C-SCPI Function Calls
- C-SCPI Preprocessor Command
- Each Supported HP VXI Register-Based Instrument

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

What Information is Provided in the Manual Pages?

The C-SCPI manual pages provide a general description of C-SCPI commands and calls. A quick reference for the supported HP register-based instruments is also provided. For additional information on these commands, calls, and HP register-based instruments, see the appropriate user's manual.

C-SCPI Macro Commands

The C-SCPI manual pages for C-SCPI macro commands provide the documentation needed to use the C-SCPI macro command. The command syntax, parameter description, and an example programming segment are all provided.

C-SCPI Function Calls

The C-SCPI manual pages for C-SCPI functions provide the documentation needed to use the functions. The parameter descriptions and an example programming segment are provided.

HP Register-Based Instruments

Each supported register-based instrument has a C-SCPI manual page that provides the SCPI command quick reference, commands not supported in C-SCPI, commands changed in C-SCPI, query command response types, a list of overlapping commands, and restrictions on the `INST_ONSRQ` command.

There is also a C-SCPI manual page called `cscpi_drivers` that provides a list of all instrument drivers available for C-SCPI. This manual page also lists what register-based instruments are supported for each C-SCPI driver. You can use the `cscpipp -?` command to find out what drivers are installed on your system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

What Does a Manual Page Look Like?

If, for example, you need information on the HP E1326B Multimeter, type the following at the user prompt:

```
man E1326B
```

The manual page for the HP E1326B Multimeter is similar to the following:

```
E1326B ()                                          E1326B ( )

NAME
    E1326B - SCPI Command Quick Reference

DESCRIPTION
    The following is a SCPI command quick reference for the
    HP E1326B Multimeter.  This quick reference provides:

    Command Descriptions

    Commands Not Supported

    Commands Changed

    Query Command Response Types

    Overlapping Commands

    ONSRQ Restrictions

    The SCPI commands are to be embedded in HP Compiled SCPI
    commands.  For information on HP Compiled SCPI, see the
    HP Compiled SCPI commands in manual pages, or see the HP
    Compiled SCPI learning products.

    For additional information on SCPI commands, see the VXI
    User's Guide for the instrument.
```

Continued on Next Page

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
SCPI COMMAND DESCRIPTIONS

    ABORt        Place multimeter in idle state.



    CALibration:LFRequency 50 | 60 | MIN | MAX
                Change line reference frequency.

ONSRQ RESTRICTIONS
    The following commands can not be used in SRQ handlers which
    execute as an interrupt routine since they internally call
    iwaithandler:

    if cscpi_overlap off

    SENS:FUNC

    SENS:[VOLT | RES]:APER

    SENS:[VOLT | RES]:NPLC

    SENS:[VOLT | RES]:RES

    CONF:....

    *RCL

    *RST

    INIT, READ?, MEAS? (for aperture = 2.5 ms) or with 1460
    scanners

    The following commands may cause an SRQ handler to execute from
    an interrupt routine if SRQ is enabled on the condition:

    Any overlapped command followed by the *OPC command with
    cscpi_overlap on.

    An INIT, READ? or MEAS? command with aperture =2.5 ms which
    sets the error bit from a timer too fast error.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notes:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

B

Compiled SCPI Software Installation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI Software Installation

The C-SCPI software is part of an entire software collection consisting of LynxOS, SICL, C-SCPI, and Free Software Foundation code. It is pre-installed on your controller at the factory

If you have had a problem with C-SCPI or have accidentally removed some of the C-SCPI files, you may be able to recover these files from the install media. To do this, connect a SCSI CD-ROM to the embedded controller, insert the install CD, and mount the file system. For example:

```
mount /dev/sd974.3 /mnt
```

Then, you can compare files in the /mnt directory with files on your hard disk. See the LynxOS manuals for more information on Lynx commands.

If you find that you need to reinstall the entire software system, please refer to the *HP E6237A VXI Pentium Real-Time Controller* manual for instructions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C

Compiled SCPI File Structure

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled SCPI File Structure

This appendix provides a description and diagrams of the directory structure for the C-SCPI software.

C-SCPI Directories

This section lists the directories and subdirectories with a brief description of each.

`/usr/hp75000`	**Main Directory for HP Compiled SCPI Software**
`/usr/hp75000/bin`	C-SCPI utilities
`/usr/hp75000/cscpi`	files used to build C-SCPI software
`/usr/hp75000/demos`	demos for HP 75000 family of products
`/usr/hp75000/demos/cscpi`	example programs used in the C-SCPI manual set

Other Directories

These are other directory that also contain C-SCPI files.

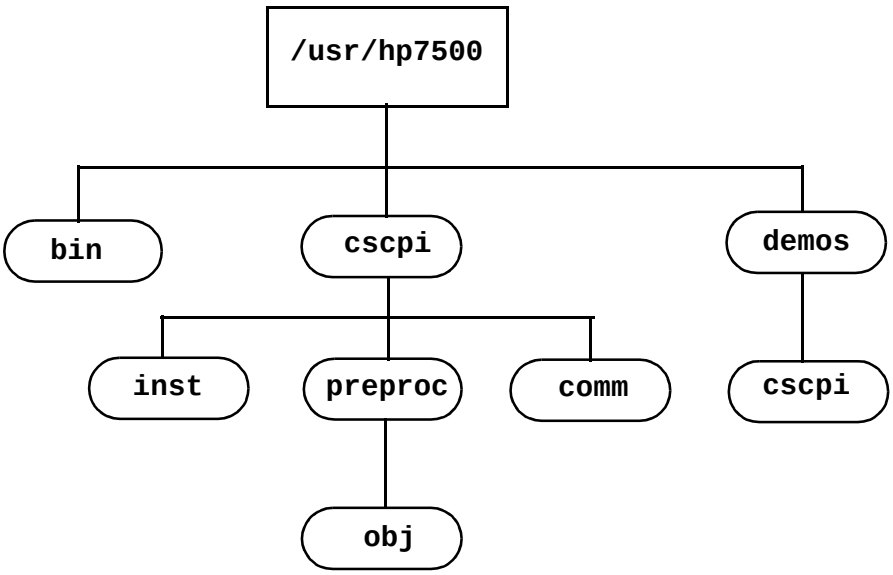
`/lib`	common library files, `libcscpi.a` is stored here
`/usr/include`	common header files, `cscpi.h` is stored here
`/usr/man`	manual pages
`/usr/man/cat1`	common manual pages, C-SCPI commands man pages are stored here
`/usr/man/cat3`	common manual pages, C-SCPI functions and library routines man pages are stored here
`/usr/man/cat5`	common manual pages, C-SCPI instrument specific details man pages are stored here

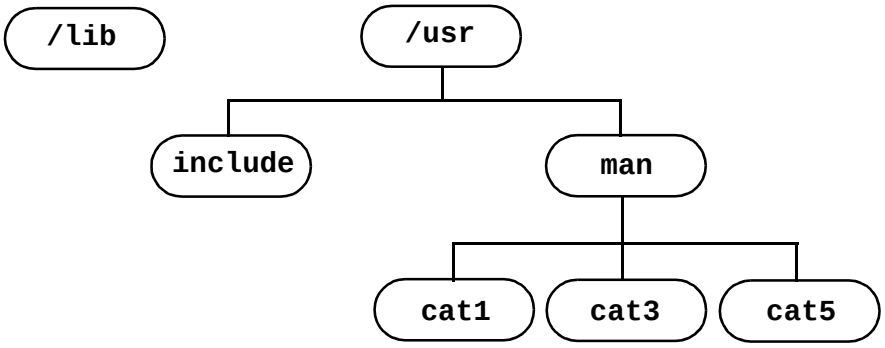
The **`/lib/thread/libcscpi.a`** file is symbolically linked to the **`/lib/libcscpi.a`** file.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Structure for C-SCPI on LynxOS

Diagram for C-SCPI Directories





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notes:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D

Threads and Compiled SCPI

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Threads and Compiled SCPI

LynxOS supports a multi-threaded processing model. Lynx uses the POSIX thread facilities that are based on the preliminary draft of POSIX 1003.4a - draft 4. See the LynxOS documentation for information on Lynx threads and using them.

Writing your Compiled SCPI Programs

When you write your C-SCPI C programs using multiple threads, the following restrictions apply:

-- `INST_STARTUP` can only be called once in your program. This C-SCPI command is used to start the register-based operating system and cannot be executed more than once.

-- `INST_OPEN` and `INST_CLOSE` must not be called from multiple threads at one time. When these C-SCPI commands are executed, the address table is modified, and you must not write to this table from multiple threads at the same time.

-- You must not access the same instrument from multiple threads at one time. Most C-SCPI commands access instruments. Some of the commands are `INST_SEND`, `INST_QUERY`, `INST_TRIGGER`, `cscpi_exe`, etc. See Chapter 5 for a complete list of C-SCPI commands.

You can protect your program against simultaneous access to one instrument from multiple threads with the LynxOS mutex library functions. See the LynxOS documentation for information on using these functions.

Compiling your Compiled SCPI Programs

The HP SICL library uses threads during its normal execution. For this reason, all C-SCPI programs are compiled with the multi-threaded option to gcc. If you use threads in your C-SCPI C programs, you do not need to do anything differently during the compile and link steps.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

E

Error Messages

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Messages

This appendix lists the possible errors you can receive while using the C-SCPI preprocessor.

Compiled SCPI Preprocessor Errors

The errors listed in the following table can be generated by the C-SCPI preprocessor. C-SCPI checks for instrument syntax errors when the C-SCPI preprocessor runs. The parameter ranges, however, are not checked until run time, which will generate a run time error.

Table E-1. C-SCPI Preprocessor Errors

Error Message	Possible Cause
`duplicate device`	The device has already been defined in an `INST_DECL`, `INST_EXTERN`, or `INST_PARAM`.
`unknown REGISTER driver`	C-SCPI does not recognize the driver parameter for a `REGISTER` configuration in the `INST_DECL`, `INST_EXTERN`, or `INST_PARAM` command. The driver may not be installed on your system, or you may be typing it in wrong.
`expected REGISTER or MESSAGE`	The *type* parameter must specify the type of configuration: `REGISTER` or `MESSAGE`.
`undeclared identifier`	The *id* that is being used was not declared in `INST_DECL`, `INST_EXTERN`, or `INST_PARAM`.
`SCPI error` <*error number*>, <*error string*>	SCPI syntax error. *error number* is the instrument error number and *error string* is the instrument error text.
`unknown driver`	The instrument used as the driver parameter when running the C-SCPI preprocessor (with -i option) is not recognized.
`use INST_QUERY for ? commands`	The *cmd_string* parameter contains a SCPI command that has a question mark (?). You must use the `INST_QUERY` command for SCPI query commands.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table E-1. C-SCPI Preprocessor Errors

Error Message	Possible Cause
`expected response format and response pointer`	When using the `INST_QUERY` command, you are missing the *readfmt* and/or *c_addr* parameters.
`expected response pointer`	When using the `INST_QUERY` or `INST_SEND`, you are missing the one of the parameters in the argument list.
`expected string`	When using `INST_QUERY` or `INST_SEND`, the *cmd_string* is missing.
`expected (`	Missing a left parentheses.
`expected )`	Missing a right parentheses.
`expected ;`	Missing a semicolon.
`expected identifier`	Missing the *id* parameter.
`expected ,`	Missing a comma in the parameter list.
`empty parameter`	Missing a parameter (, ,).
`missing or extra”`	Missing or extra double quote.
`missing or extra ‘`	Missing or extra single quote.
`usage: cscpipp [-i driver] [-f datafile] [file]`	The correct usage of the preprocessor is as stated.
`only 1 file allowed`	You cannot specify more than one file for the preprocessor.

Compile and Link Errors

Compile and Link errors can occur because of problems in your C-SCPI program. See Chapter 4, “Troubleshooting Compiled SCPI” for specific information on resolving compile and link errors.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Run-Time Errors

Run time errors occur when running your executable code. These errors can occur for various reasons. You can include a `cscpi_error` function to trap instrument run time errors. You can also check the `cscpi_open_error` global variable to see what type of open errors occur. See Chapter 4, "Troubleshooting Compiled SCPI" for more information on these error checking techniques.

Table E-2. cscpi_open_error Descriptions

Error #	Most Likely Cause	Description of Cause
0	No error has occurred.	No error has occurred.
1	`INST_STARTUP` was **not** included in the program before `INST_OPEN`.	See "Resolving Compiled SCPI Run-Time Errors" beginning on page 56.
2	A mismatch between the declaration and the open for the instrument(s) was detected by the instrument driver.	This occurs because the declaration made for the instrument (`INST_DECL`) did not match the instrument that was opened in the `INST_OPEN` command. This could also occur if one of the cards in a scanning multimeter is not supported.
3	System is out of memory.	Check your system's resources.
4	Format of the address encountered with a multiple card instrument was incorrect. Format: `INST_OPEN(vm,"vxi,(nn,nn)");`	This occurs when the software cannot understand an address of a multiple card instrument (scanning multimeter or Digital Functional Test System, for example).
5	Invalid address was encountered (SICL `iopen("vxi,nn")` call failed.)	See **Program Description 2** of this section.
6	SICL is not setup properly or not running (SICL `iopen("vxi")` call failed).	Determine if SICL is running on your system. (See **Other Causes** of **Program Description 2** of this section.)
7 or 8	SICL has encountered a resource problem, or an Internal SICL error has occurred.	Contact your local Support organization.
9, 10, or 11	Internal SICL error has occurred.	Contact your local Support organization.
12 or 13	System encountered a resource problem	Contact your local Support organization.
14	Instrument driver cannot provide the required information.	This occurs when the instrument driver is not compatible with the version of C-SCPI that is installed on your system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

F

Other Documentation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Other Documentation

The C-SCPI manual set in itself does not provide you with all the information needed to successfully use this product. The following list might help you find additional information:

- HP Pentium Controller Learning Products
- Lynx Real-Time Operating System Learning Products
- HP VXI Instrument Manuals

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A

Accessing C-SCPI Online Documentation, 10
Advantages
  Compiled SCPI, 12
  cscpi_error routine, 81
  interactive functions, 32
  overlapped mode, 36
ANSI C, 6, 11

B

Block Data, storing, 27–28, 59–61

C

-c Option, 7, 22
C Program, getstrt1.cs, 9
Clear Instrument Command, 88–89
Close Instrument Command, 90–91
Comma Operator
  INST_QUERY, 103
  INST_SEND, 111
Command Reference, 86–129
Commands
  *OPC?, 44, 46
  *WAI, 46
  -c option, 7, 22
  cscpi_datafile, 28
  cscpi_error, 50, 81–83, 119
  cscpi_exe, 32, 62–63, 120–121
  cscpi_exe_fildes, 33, 122–123
  cscpi_exe_stream, 33, 124–125
  cscpi_get_overlap, 40, 126
  cscpi_overlap, 40–42, 127
  cscpipp - ?, 10, 133
  -f option, 27–28, 59–61
  -g option, 7, 22
  GDB software tool, 79
  -I option, 7, 22
  -i option, 29–31
  iintrof, 44
  iintron, 44
  INST_CLEAR, 88–89
  INST_CLOSE, 90–91
  INST_DECL, 92–93
Commands (*continued*)
  INST_EXTERN, 53–54, 94–95
  INST_ONSRQ, 96–97
  INST_OPEN, 98–99
  INST_PARAM, 56, 100–101
  INST_QUERY, 102–107
  INST_READSTB, 108–109
  INST_SEND, 28, 110–115
  INST_STARTUP, 116
  INST_TRIGGER, 117
  -l option, 23
  make, 25–26
  man, 10, 132–135
  -mthreads option, 7, 22
  non-overlapping, 40
  -o option, 23
  overlapped, 40
  preprocessor, 27
  quick reference, 128–129
  SCPI only files, 29–31
Compiled SCPI
  advantages of using, 12
  command reference, 86–129
  description, 13
  directories, 140
  errors, 146, 148
  file structure, 140–141
  getting started with, 2–14
  interactive functions, 32–35, 62–63
  learning about, 11–14
  overlapped mode, 36–46
  overview, 16–17
  preprocessor command, 27
  programming with, 48–64
  running first program, 6–10
  software installation, 138
  structure on LynxOS, 141
  threads, 144
  throughput, 12, 14
  troubleshooting, 66–83
  using, 16–46
    GDB software, 79–80
    in interactive mode, 62–63

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C (*continued*)

Commands (*continued*)
  verifying
    system setup, 4–5
    your system, 3–5
  who should use, 12
  writing programs, 144
Compiling
  -c option, 22
  errors, resolving, 69–72
  -g option, 7, 22
  -I option, 22
  linking source code, 22–24
  -mthreads option, 22
  programs, 6, 144
Configuration
  examples, 48
  requiring overlapped mode, 37
Controlling Overlapped Execution, 43
Conventions Used, 2
Creating Source Code, 18
C-SCPI
  creating executable code, 6
  directories, 140
  file structure, 140–141
  learning about, 11–14
  library, 24
  online documentation, accessing, 10
  overview, 16–17
  preprocessor
    cscpipp, 20, 66
    errors, 66–68
    running the, 20–21
  process, 8
  software installation, 138
  source code, 18, 22–24
  structure on LynxOS, 141
  threads, 144
  using, 16–46
    in interactive mode, 62–63
  verifying
    system setup, 4–5
    your system, 3–5
  writing programs, 144
C-SCPI Commands
  command reference, 86–129
  defining, 19
  get overlap function, 126
  header file, 19
  instrument
    clear, 88–89
    close, 90–91
    declaration, 19, 92–93
    execution, 120–125
    external, 94–95
    initialization, 19
    open, 98–99
    parameter, 100–101
    programming commands, 19
    query, 102–107
    read status byte, 108–109
    run-time errors, 119
    send, 110–115
    service request, 96–97
    startup, 116
    trigger, 117
  macro commands, 87–117
    manual pages, 133
  overlap function, 127
  quick reference, 128–129
C-SCPI Functions, 118- 129
  cscpi_error, 119
  cscpi_exe, 120–121
  cscpi_exe_fildes, 122–123
  cscpi_exe_stream, 124–125
  cscpi_get_overlap, 126
  cscpi_overlap, 127
  manual pages, 133
cscpi Library, 24
cscpi.h Header File, 19
cscpi_datafile, 28
cscpi_drivers Manual Page, 133
cscpi_error Command, 50, 81–83, 119
  advantages of using, 81
  example, 83
  linking, 50
cscpi_exe Command, 32, 62–63, 120–121
cscpi_exe_fildes Command, 33, 122–123

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C (*continued*)

cscpi_exe_stream Command, 33, 124–125
cscpi_get_overlap Command, 40, 126
cscpi_open_error, 74–77, 148
cscpi_overlap Command, 40–42, 127
cscpipp, 6
cscpipp - ? Command, 10, 133
cscpipp Command, 20, 66

D

Debugging, 32
  using GNU debugger, 79–80
Defining C-SCPI Commands
  header file, 19
  instrument
    declaration, 19
    initialization, 19
    programming commands, 19
Determining Command Order
  overlapped mode, 45
Disabling Interrupts, 44

E

Embedded Controller, 13
  configuration, 49
  external triggering, 64
Enabling Interrupts, 44
Errors
  compile and link errors, resolving, 69–73
  compile errors, 69–72
  error messages, 146, 148
  link errors, 69, 73
  preprocessor errors, resolving, 66–68
  run-time errors, 148
    description, 82
    resolving, 74–78
  syntax errors, 66
  trapping with cscpi_error, 50, 81–83, 119
  troubleshooting, 66–83
  usage errors, 67–68
Example Program
  error routines, 50
  getstrt1.cs, 6, 9
Example Program (*continued*)
  programming
    with external files, 53–55
    with parameter list, 56–58
    with scanning multimeter, 51–52
  using
    -f option, 28, 59–61
    -i option, 30
    interactive mode, 62–63
Examples
  compile error, 69–72
  directory location, 48
  link error, 73
  programming
    with a parameter list, 56–58
    with a scanning multimeter, 51–52
    with an external file, 53–55
  run-time errors, 74–78, 82
  syntax error, 66
  system configuration, 48
  usage error, 67–68
  with cscpi_error, 50
Executable Code, creating, 8
Executing GDB, 80
External
  file, programming with, 53–55
  triggering with embedded computer, 64
  variable reference, 94–95

F

-f Option, 27–28, 59–61
File Descriptors, 33, 122
File Structure, 140–141
Format Specifiers, 103–105, 111–113
  %a, 105, 113
  %b, 104, 113
  %d, 103, 111
  %f, 104, 111
  %r, 104, 112
  %S, 104, 112
  %s, 104, 112

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

F (*continued*)

Function Calls, 118–127, 129, 133
  cscpi_error, 119
  cscpi_exe, 32, 62–63, 120–121
  cscpi_exe_fildes, 33, 122–123
  cscpi_exe_stream, 33, 124–125
  cscpi_get_overlap, 40, 126
  cscpi_overlap, 40–42, 127

G

-g Option, 7, 22
GDB
  commands, 79
  executing, 80
  software tool, 79–80
getstrt1.cs Example Program, 9
Getting Started with Compiled SCPI, 2–14
  Compiled SCPI, 6
GNU Debugger, 79–80
Group Execute Trigger, 117

H

Header File, 19
Help
  error messages, 146, 148
  online help, 132–135

I

-I Option, 7, 22
-i Option, 29–31
iintroff Command, 44
iintron Command, 44
INST_CLEAR Command, 88–89
INST_CLOSE Command, 90–91
INST_DECL Command, 92–93
INST_EXTERN Command, 53–54, 94–95
INST_ONSRQ Command, 96–97
INST_OPEN Command, 98–99
INST_PARAM Command, 56, 100–101
INST_QUERY Command, 102–107
INST_READSTB Command, 108–109
INST_SEND Command, 28, 110–115
INST_STARTUP Command, 116
INST_TRIGGER Command, 117
Installing Software, 138
Instrument
  clear command, 88–89
  close command, 90–91
  declaration, 19
    INST_DECL, 92–93
  external, INST_EXTERN, 94–95
  initialization, 19
  open command, 98–99
  parameter command, 100–101
  programming commands, 19
  query command, 102–107
  read status byte command, 108–109
  run-time errors, 82, 119
  send command, 110–115
  service request command, 96–97
  startup command, 116
  trigger command, 117
Interactive Functions, 32–35, 62–63
  advantages, 32
  debugging, 32
  error checking, 32
  programming with, 34–35, 62–63
  using, 32–33
Interrupts, 43

L

-l Option, 23
Learning About Compiled SCPI, 11–14
Library, 24
  cscpi, 24
  m (math), 25
  sicl, 24
Linking
  cscpi library, 23
  errors, resolving, 69, 73
  -g option, 7
  -l option, 7, 23
  -o option, 7, 23
  programs, 7
  source code, 22–24
LynxOS File Structure, 141

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

M

Macro Commands, 87–117, 133
  INST_CLEAR, 88–89
  INST_CLOSE, 90–91
  INST_DECL, 92–93
  INST_EXTERN, 94–95
  INST_ONSRQ, 96–97
  INST_OPEN, 98–99
  INST_PARAM, 100–101
  INST_QUERY, 102–107
  INST_READSTB, 108–109
  INST_SEND, 110–115
  INST_STARTUP, 116
  INST_TRIGGER, 117
Make Command, 25–26
Makefiles, 25–26
man Command, 10, 132–135
Manual Pages, 132–135
  appearance, 134–135
  C-SCPI function calls, 133
  C-SCPI macro commands, 133
  cscpi_drivers, 133
  how to use, 132–135
  register-based instruments, 133
MESSAGE Configuration
  INST_DECL, 92–93
  INST_EXTERN, 94–95
  INST_PARAM, 100–101
-mthreads Option, 7, 22
Multimeter, programming with, 51–52

N

Non-Overlapping Command, 40

O

-o Option, 23
Online Documentation, 132–135
Online Help, 132–135
*OPC?, 44, 46
Open Instrument Command, 98–99
Other Documentation, 150
Overlapped Commands, 40

O (*continued*)

Overlapped Mode, 36–46
  completing system calls, 44
  configurations requiring, 37
  controlling overlapped execution, 43
  cscpi_get_overlap, 40, 126
  cscpi_overlap, 40–42, 127
  default mode, 36
  determining command order, 45
  determining to use, 36
  overlapped commands, 40
  parallel commands, 36
  programming for efficiency, 45–46
  throughput, 37–39
  using, 40, 42
    *OPC?, 44, 46

P

Parameter List, programming with, 56–58
Preprocessor
  command, 27
  error messages, 146, 148
  errors, resolving, 66–68
  -f option, 27–28, 59–61
  options, 27–31
  running the, 20–21
Programming
  for efficiency, overlapped mode, 45–46
  with a parameter list, 56–58
  with a scanning multimeter, 51–52
  with an external file, 53–55
  with compiled SCPI, 48–64
  with interactive functions, 34–35, 62–63
  writing programs, 144
Protecting System Calls
  iintroff, 44
  iintron, 44
  using *OPC?, 44

Q

Query Instrument Command, 102–107
Quick Reference, 128–129

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

R

Read Status Byte Command, 108–109
REGISTER Configuration
  INST_DECL, 92–93
  INST_EXTERN, 94–95
  INST_PARAM, 100–101
Resolving
  compile and link errors, 69–73
  compiled SCPI preprocessor errors, 66–68
  run-time errors, 74–78
Reviewing
  C-SCPI process, 8
  getstrt1.cs program, 9
Running
  C-SCPI preprocessor, 6, 20–21
    cscpipp, 6, 66
  first C-SCPI Program, 6–10
running first program, 6
Run-time
  commands, 32
  errors, 74–78, 82, 119, 148

S

Scanning Multimeter, programming with, 51–52
SCPI Commands
  using, 29–31
SCPI Description, 12
Send Instrument Command, 110–115
Sequential
  commands, 38–39
  programming, 36
Service Request Command, 96–97
SICL
  description, 12
  library, 24
  more about, 13
Software Installation, 138
Source Code
  compiling/linking, 22–24
  creating, 18
Start Instrument Command, 116
Storing Block Data, 27–28, 59–61
Streams, 33, 124–125
Syntax Errors, 66
System Configurations, 48

T

Threads and C-SCPI, 144
Throughput
  comparing two programs, 38–39
  Compiled SCPI, 12, 14
  message-based cards, 12, 14
  order of commands, 45
  overlapped mode, 37–39
  register-based cards, 12, 14
Time
  for overlapped programs, 39
  for sequential programs, 39
Trapping Errors, 50, 81–83
Trigger Instrument Command, 117
Triggering with HP Embedded Computer, 64
Troubleshooting, 66–83
Tutorial, 6

U

Usage Errors, 67–68
Using
  *OPC?, 44, 46
  Compiled SCPI, 16–46
    in interactive mode, 62–63
  C-SCPI, 16–46
    in interactive mode, 62–63
  cscpi_error, 83
  GDB software, 79–80
  GNU debugger, 79–80
  interactive functions, 32–33
  libraries, 24
  makefiles, 25–26
  overlapped mode, 40, 42
  SCPI only files, 29–31

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

V

Verifying
  C-SCPI system, 3–5
  system setup, 4–5
VXI
  controller, embedded, 13, 49
  description, 12
  instrument run-time errors, 82
  instruments, 13

W

*WAI, 46
Wait for Complete (*WAI), 46
Writing C- SCPI Programs, 144

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP