Teradyne ICT Application Brief	Teradyne, Inc www.teradyne.com	Author: Bobby Griffis Published Date: May 2012

Dynamic Programming Extension… what is it and what can I use it for?

Support for DLL communication on the TestStation platform

Application Overview

The TestStation system has for many years been a platform that companies have used in their production environment to test the quality of their products. During this time TestStation has continued to evolve to meet the needs of our customers. An increasingly requested requirement is the ability to interact with external test instruments and software in a fast, efficient and standardized manner. This has lead to the development of the Dynamic Programming Extension option.

The Dynamic Programming Extension may be used to easily and quickly pass test program status and variables to and from any software application that was written in Microsoft Visual Studio C/C++ or LabWindows/CVI. These features simplify the programming activities that typically must be performed when you want to control external instruments or communicate with an external software application during test program runtime execution.

This application brief provides an introduction to the Dynamic Programming Extension and, as an example, how it may be used to control a Hewlett Packard DMM through TPG initiated DLL communications.

Hardware/Software Requirements

The application outlined in this document is focused on a virtual software integration in order to illustrate the software components, so no test equipment hardware is required.

The user needs a PC with the following software: TestStation Software version 6.4.0 Patch 20 or higher, Labwindows/CVI, and Hewlett Packard 34401A DMM driver.

If this example were to be extended to communicate with a third party instrument connected to the test system the most common implementation would be with a PXI instrument. In this case, the application would require a TestStation UltraPin test system equipped with an Integrated Controller, Accessory Slot w/TestStation PXI Expansion board, Dynamic Programming Extension License, the target PXI instrument(s) with integration cables and the appropriate hardware drivers for the PXI instrument(s).

Application Description

The TestStation test program may contain the Runtime DLL System Subroutines or calls to the functions

inside a DLL. The DLLs contain functions to obtain data and send it back to the test program. The TestStationCLIB library contains functions specifically designed to allow communication between DLLs and test programs. A list of these functions is shown in Figure 1.

Figure 1. DLL Extensions to Navigate Runtime

TEST_STATION_INIT()	Starts Communication
TEST_STATION_Exit()	Stops Communication
TEST_STATION_AddHandler()	Add Call Back Function
TEST_STATION_RemoveHandler()	Remove Call Back Function
TEST_STATION_PrintToMESdevice()	Send message to message output
TEST_STATION_PrintToOutputWindow()	Send message to Navigate window
TEST_STATION_PringStringtoMESdevice()	Send message to runtime message output
TEST_STATION_PrintStringToOutputWindow()	Send message to Navigate window
TEST_STATION_SetFloatVariable()	Modifies a global FLOAT variable in the tpg
TEST_STATION_GetFloatVariable()	Gets the value of a FLOAT variable from the tpg
TEST_STATION_SetCSTRINGVariable()	Modifies CSTRING variable in the tpg
TEST_STATION_GetCSTRINGVariable()	Gets the value of a CSTRING variable from the tpg
TEST_STATION_SetBSTRINGVariable_BitValue()	Modifies BSTRING variable in the tpg
TEST_STATION_GetBSTRINGVariable_BitValue()	Gets the value of a BSTRING variable from the tpg
TEST_STATION_GetDirectory()	Retrieves directory path
TEST_STATION_AbortRTS()	Start the abort process in the runtime
TEST_STATION_ResetRTS()	Start the reset process in the runtime
TEST_STATION_GetVersion()	Gets the current version of the TestStation Library
TEST_STATION_SetTimeout()	Sets the time for the runtime to wait for the dll process
TEST_STATION_GetLabel()	Get the current test label
TEST_STATION_SetFailBit()	Sets a bit in the fail string
TEST_STATION_LOG_Measurement()	Allows data to be sent directly to the log file
TEST_STATION_GetRtsOptions String()	Retrieves the string value of an rtsOption
TEST_STATION_GetRtsOptionsBool()	Retrieves a Boolean value from the rtsOptions
TEST_STATION_GetRtsOptionsInt()	Retrieves an integer from the rtsOptions

Using the TestStationCLIB functions, you can access TestStation language variables such as FAIL using DLLs. DLLs are written in the C/C++ or the LabWindows/CVI language then connected, linked, and installed on the tester. TestStation Software supports Microsoft Visual MSVC 6 or LabWindows/CVI 7.0.

A mechanism for making calls to Runtime DLL functions was incorporated in the test language. After a Runtime DLL is made available on the test system, the test programmer only needs to know the DLL name and its calling syntax.

When using Runtime System DLLs, no external code is included in the compiled test program. Instead, when a DLL function is called, a CALL statement issues a series of instructions to the Runtime System to load the DLL and communicate with it.

The examples below show how the DLL functions might be used in an external DLL, how external DLLs would be declared and functions called in the TestStation test program and how the functions may write messages to the TestStation Runtime output.

DLL with functions to communicate with virtual DMM

```
/*
 * cvi example using virtual dmm driver
 */

#include <ansi_c.h>
```

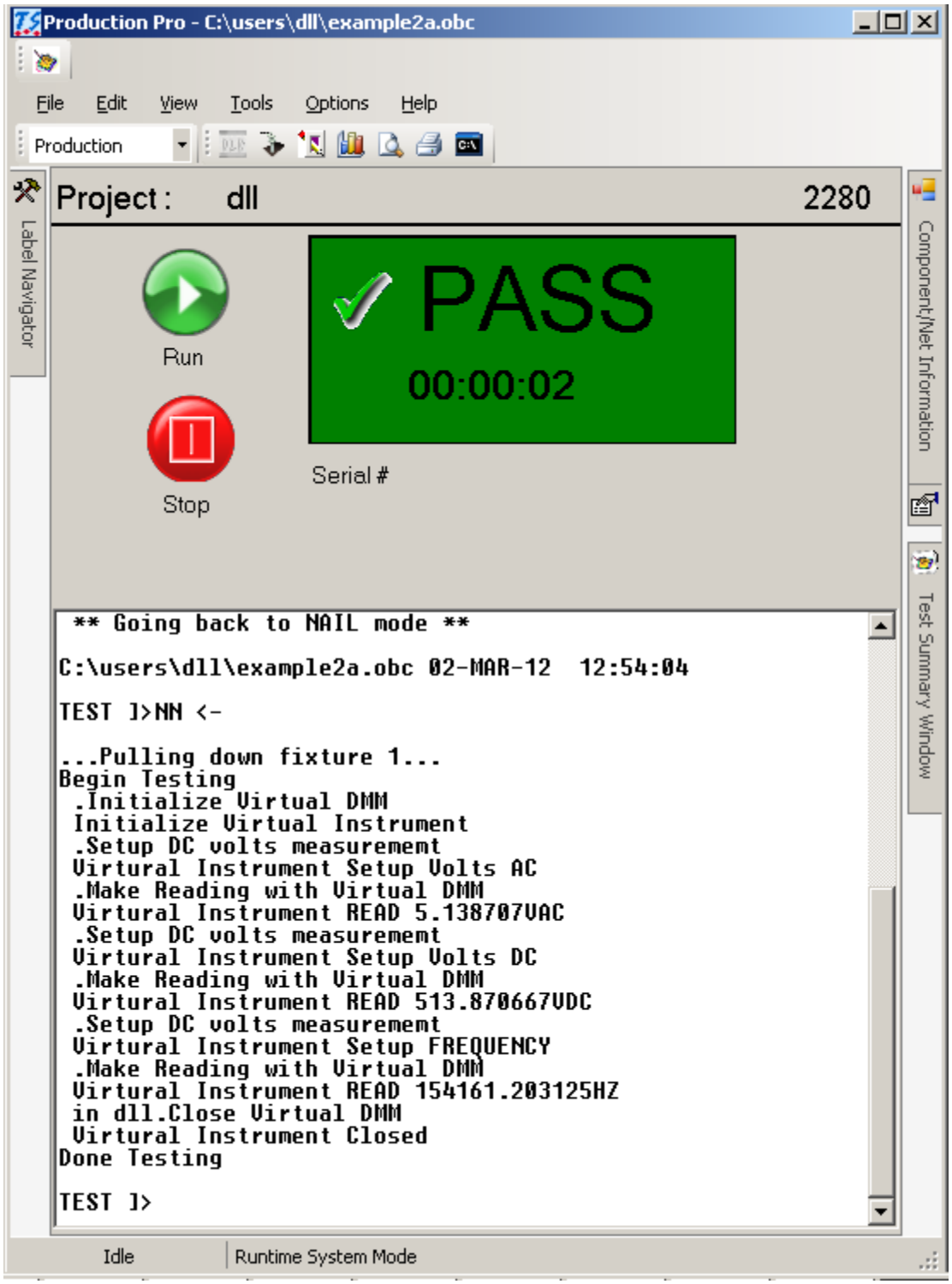
```
#include "hp34401a.h"
#include "TestStationclib.h"
#define FALSE 0
#define TRUE 1

static ViReal64 reading;
static ViSession hndl;
char dllname[] = "example2";

int __stdcall DllMain(HINSTANCE hint, DWORD dwReason, LPVOID pvReserved){

     int stat = FALSE;
   switch(dwReason){
       case DLL_PROCESS_ATTACH:
       stat = TEST_STATION_Init(dllname,1);
       break;

       case DLL_PROCESS_DETACH:
       stat = TEST_STATION_Exit(dllname);
       break;

       default:
       break;
 }
return TRUE;

}

 /*
  * external exported function calls
  */
extern __declspec(dllexport) init(){

     TEST_STATION_PrintToOutputWindow(" .Initialize Virtual DMM\n");
     hp34401a_InitWithOptions("GPIB0::23::INSTR",VI_FALSE,VI_TRUE,
       "Simulate=1", &hndl);
 return 0;
}


extern __declspec(dllexport) setup(char *set){

     TEST_STATION_PrintToOutputWindow(" .Setup DC volts measurememt\n");
     if(strcmp(set,"AC") == NULL)
       hp34401a_ConfigureMeasurement(hndl,HP34401A_VAL_AC_VOLTS,1.5,0.001);
     if(strcmp(set,"DC") == NULL)

hp34401a_ConfigureMeasurement(hndl,HP34401A_VAL_DC_VOLTS,HP34401A_VAL_AUTO_RANGE_ON,0.001)
;
     if(strcmp(set,"FREQ") == NULL)

hp34401a_ConfigureMeasurement(hndl,HP34401A_VAL_FREQ,HP34401A_VAL_AUTO_RANGE_ON,0.001);
 return 0;
}

extern __declspec(dllexport) read(float *val){

     TEST_STATION_PrintToOutputWindow(" .Make Reading with Virtual DMM\n");
     hp34401a_ConfigureTrigger(hndl,HP34401A_VAL_IMMEDIATE,0.01);
     hp34401a_Read(hndl,1000,&reading);
     *val = (float) reading;
 return 0;
}

extern __declspec(dllexport) close(){
     TEST_STATION_PrintToOutputWindow(" in dll.Close Virtual DMM\n");
     hp34401a_close(hndl);
 return 0;
}
```

Then with this DLL you can structure the test program to communicate to the DMM and retrieve test results.

TPG utilizing calls to DLL functions

```
DECLARE BSTRING FAIL(10);

DECLARE EXTERNAL TPGEXAMPLE2a() FUNCTION 'init@example2a.dll';
DECLARE EXTERNAL TPGEXAMPLE2b(CSTRING TYP(4)) FUNCTION 'setup@example2a.dll';
DECLARE EXTERNAL TPGEXAMPLE2c(FLOAT VAL) FUNCTION 'read@example2a.dll';
DECLARE EXTERNAL TPGEXAMPLE2d() FUNCTION 'close@example2a.dll';

DECLARE GLOBAL RVAL;

ENDDEC:


WRITE 'Begin Testing%NL%';

INIT:
   CALL TPGEXAMPLE2a();
   WRITE ' Initialize Virtual Instrument%NL%';

SETUPR1:
   CALL TPGEXAMPLE2b(TYP='AC');
   WRITE ' Virtural Instrument Setup Volts AC%NL%';

   CALL TPGEXAMPLE2c(VAL=RVAL);
   WRITE ' Virtural Instrument READ %F%VAC%NL%'RVAL;

SETUPR2:
   CALL TPGEXAMPLE2b(TYP='DC');
   WRITE ' Virtural Instrument Setup Volts DC%NL%';

   CALL TPGEXAMPLE2c(VAL=RVAL);
   WRITE ' Virtural Instrument READ %F%VDC%NL%'RVAL;

SETUPR3:
   CALL TPGEXAMPLE2b(TYP='FREQ');
   WRITE ' Virtural Instrument Setup FREQUENCY%NL%';

   CALL TPGEXAMPLE2c(VAL=RVAL);
   WRITE ' Virtural Instrument READ %F%HZ%NL%'RVAL;

CLOSE:
   CALL TPGEXAMPLE2d();
   WRITE ' Virtural Instrument Closed%NL%';

WRITE 'Done Testing%NL%';


END;
```

Using TestStation DLL extension functions, the DLL may send messages back to the Runtime. Below is an example of sending messages to the Runtime standard output.


Production Pro - C:\users\dll\example2a.obc
File Edit View Tools Options Help
Production
Project : dll 2280
Label Navigator
Component/Net Information
Test Summary Window
Run
Stop
PASS
00:00:02
Serial #
** Going back to NAIL mode **
C:\users\dll\example2a.obc 02-MAR-12 12:54:04
TEST]>NN <-
...Pulling down fixture 1...
Begin Testing
.Initialize Virtual DMM
Initialize Virtual Instrument
.Setup DC volts measurememt
Virtural Instrument Setup Volts AC
.Make Reading with Virtual DMM
Virtural Instrument READ 5.138707VAC
.Setup DC volts measurememt
Virtural Instrument Setup Volts DC
.Make Reading with Virtual DMM
Virtural Instrument READ 513.870667VDC
.Setup DC volts measurememt
Virtural Instrument Setup FREQUENCY
.Make Reading with Virtual DMM
Virtural Instrument READ 154161.203125HZ
in dll.Close Virtual DMM
Virtural Instrument Closed
Done Testing
TEST]>
Idle
Runtime System Mode

Additional Information

Additional information on the Dynamic Programming Extension (DLL support) option and the TestStation Functional Expansion Board may be found at the Teradyne website www.teradyne.com or by contacting your Teradyne representative.

For detailed documentation on the Dynamic Programming Extension (DLL support) option and the TestStation Functional Expansion Board please refer to the TestStation Test Language Reference Manual, TestStation PXI Expansion Board User's Guide, and various Applications notes, all which can be found in Teradyne's eKnowledge support site.