Qualcomm Technologies International, Ltd.

Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

CSR

Push every boundary.™

BlueCore ®

TrueTest® Toolkit Introduction

Application Note

Issue 11

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Document History

Revision	Date	Change Reason
Draft A	05 JUL 04	Original publication of this document. (CSR reference: bcore-an-051Pa)
Draft B	08 AUG 06	Updated for latest DLLs and wrappers
Draft C	05 SEP 06	Updated example application code
4	31 OCT 06	Updated with details of VB2005 API file and unsigned parameter issues with VB6/7. Corrected details regarding installation. (CSR reference: CS-101531-ANP4)
5	06 FEB 07	Added LabView examples
6	24 SEP 07	Removed references to /unsafe compiler option for C# apps. Updated the application code and LabView examples
7	26 SEP 07	Update to application code example
8	11 FEB 11	Updates for BlueSuite 2.4
9	13 SEP 12	Updates for BlueSuite2.5 and to latest CSR style
10	26 SEP 14	Minor updates and updated to latest CSR style
11	28 APR 15	Minor updates for BlueSuite 2.6 and latest template

Trademarks, Patents and Licences

Life Support Policy and Use in Safety-critical Compliance

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

Contents

Contacts 2
1 TrueTest Overview 6
1.1 TestEngine 6
1.2 TestFlash 6
1.3 TestE2 7
1.4 Installation Files 7
2 Installing TrueTest 8
3 Using TrueTest 9
3.1 General Points 9
3.1.1 Building a C/C++ Application Using Borland 9
3.1.2 Visual Basic and Unsigned Function Arguments 9
3.1.3 Execution Path 9
3.2 Using the TestEngine API 10
3.2.1 Building a C/C++ Application 10
3.2.2 Building a Visual BasicApplication 10
3.2.3 Building a Delphi Application 10
3.2.4 Building a C# Application 10
3.2.5 Example Application 10
3.3 Using the TestFlash API 11
3.3.1 Building a C/C++ Application 11
3.3.2 Building a Visual Basic Application 11
3.3.3 Building a Delphi Application 11
3.3.4 Building a C# Application 11
3.3.5 Example Application 12
3.4 Using the TestE2 API 12
3.4.1 Building a C/C++ Application 12
3.4.2 Building a Visual Basic Application 13
3.4.3 Building a Delphi Application 13
3.4.4 Building a C# Application 13
3.4.5 Example Application 13
3.5 Using LabView 14
3.5.1 Introduction 14
3.5.2 Basic Function Calling 14
3.5.3 Reading Values Using Pointers 16
3.5.4 Passing a String to a LabView DLL Function 17
3.5.5 OpenTestEngine, OpenTestEngineSpi and CloseTestEngine 18
4 Terms and Definitions 20

List of Figures

Figure 2.1 TrueTest Directory Structure 8

1 TrueTest Overview

The HCI, radiotest, Persistent Store (PS), Flash and EEPROM software tools released with BlueSuite (BTCli, BlueTest3, PSTool, BlueFlash, E2CMD) are generally used during development and design validation. These programs are not intended for use during production testing, as they require multiple button presses or manual data entry that may lead to operator error in a busy production environment.

The TrueTest toolkit contains a set of library files that allow CSR's customers access to BlueCore test and configuration functionality using programmer APIs. The APIs are very simple and support most programming languages, including Microsoft Visual Basic (VB 6, 7 and 8 (2005) or later), C/C++, C#, Delphi and LabView.

This document is an introduction to the TrueTest libraries. Each library is released with its own compiled HTML (`.chm`) help file, which provides further user information, including the API reference.

1.1 TestEngine

TestEngine allows a test application to be developed that can communicate with one or more BlueCore devices from the same PC.

The TestEngine library supports access to BlueCore using:

- UART, i.e. BCSP, H4, H4DS, and H5 protocols
- USB
- SPI, i.e. LPT or USB-SPI

TestEngine allows the programmer to control various test modes and to configure each device. TestEngine supports various command sets such as:

- HCI, Device Danager (DM)
- BlueCore Commands (BCCMDs)
- Host query commands (HQCMDs)

The SPI interface is limited in the command sets it supports and does not support HCI or DM. Support for HQCMD over SPI is present in firmware version 23 and later. It may also be the case that some BlueCore firmware versions do not support certain command sets. See the relevant firmware release note for confirmation of command set support.

TestEngine is comprised of the following files:

- `TestEngine.dll`: The dynamically linked library file
- `TestEngine.lib`: The library file required for linking with certain programming languages
- `TestEngine.h`: C header file containing function declarations
- `TestEngine.chm`: Help file containing function descriptions and example code
- `TestEngineAPI.bas`: VB6 file containing function declarations
- `TestEngineAPI.vb`: VB7 file containing function declarations
- `TestEngineAPI_05.vb`: VB 2005 file containing function declarations
- `TestEngineAPI.pas`: Delphi file containing function declarations
- `TestEngineAPI.cs`: C# class wrapper file for TestEngine functions

1.2 TestFlash

TestFlash allows a programming utility to be developed.

The TestFlash library supports flash programming over the SPI interface. The library can be used to program:

- Single devices
- Multiple devices using multiple connections
- Up to sixteen devices using the CSR designed gang programmer (DEV-PC-1140B)

TestFlash includes the following files:

- `TestFlash.dll`: The dynamically linked library file
- `TestFlash.lib`: The library file required for linking with certain programming languages
- `TestFlash.h`: C header file containing function declarations
- `TestFlash.chm`: Help file containing function descriptions and example code
- `TestFlashAPI.bas`: VB6 file containing function declarations
- `TestFlashAPI.vb`: VB7 file containing function declarations
- `TestFlashAPI_05.vb`: VB 8 (2005) file containing function declarations
- `TestFlashAPI.pas`: Delphi file containing function declarations
- `TestFlashAPI.cs`: C# class wrapper file for TestFlash functions

1.3 TestE2

TestE2 allows an EEPROM programming utility to be developed.

The TestE2 library supports full manipulation of an EEPROM connected to a BlueCore IC using the SPI interface.

It supports:

- EEPROM configuration
- Clearing and programming of single devices or up to sixteen devices attached to the CSR designed gang programmer (DEVPC-1140B)

TestE2 is the replacement for the E2Api library, which is no longer released.

TestE2 includes the following files:

- `TestE2.dll`: The dynamically linked library file
- `TestE2.lib`: The library file required for linking with certain programming languages
- `TestE2.h`: C Header file containing function declarations
- `TestE2.chm`: Help file containing function descriptions and example code
- `TestE2API.bas`: VB6 file containing function declarations
- `TestE2API.vb`: VB7 file containing function declarations
- `TestE2API_05.vb`: VB 2005 file containing function declarations
- `TestE2API.pas`: Delphi file containing function declarations
- `TestE2API.cs`: C# class wrapper file for TestE2 functions

1.4 Installation Files

The TrueTest toolkit is released as part of BlueSuite, and is therefore installed by the BlueSuite installer. The required drivers for SPI and the host transports as well as support libraries used by the TrueTest libraries are installed as part of the BlueSuite installation.

2 Installing TrueTest

The TrueTest toolkit is installed as part of BlueSuite, shows the directory structure:

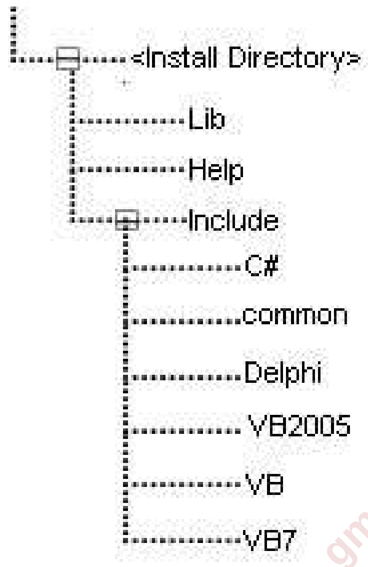


Figure 2.1: TrueTest Directory Structure

Note:

When building a LabView executable, the DLLs should be included in the data directory that resides in the same location as the executable.

3 Using TrueTest

3.1 General Points

3.1.1 Building a C/C++ Application Using Borland

The supplied import libraries for the DLLs are in COFF format. Borland users should use `implib.exe` to extract an OMF import library from the DLL.

3.1.2 Visual Basic and Unsigned Function Arguments

Some of the functions in the TrueTest APIs take arguments of type `uint16` or `uint32` or return values of these types. These are unsigned 16 or 32-bit values. While Visual Basic 8 (`2005 .NET`) and later supports these types, Visual Basic versions 6 and 7(`.NET`) do not have any equivalent unsigned data types, so it is necessary to work around the limitation in code.

This section describes a workaround for this issue, using the `uint16` type as an example. The same principle can be applied for `uint32` values, but with different values for the limits of 32-bit signed / unsigned types. The workaround is not required for VB2005 (`.NET`) and later as the new unsigned types (such as `UShort` and `UInteger`) are used in the VB2005 API file.

Arguments of type `uint16` are represented in Visual Basic as the signed Short data type, which is 16-bits wide but represents the values -32768 to 32767. This presents a problem when trying to pass values greater than 32767 to functions that take `uint16` (Short) arguments because the Visual Basic compiler does not accept these values as valid for that type. Therefore, unsigned values greater than 32767 must be represented by their equivalent negative value.

For example, to pass the unsigned value 32768 as a `Short`, the value -32768 must be passed and to pass the unsigned value 65535, the value -1 must be passed.

The equivalent `Short` value can be found as shown in this example which converts the required unsigned value represented as a 32-bit Integer to itsShort equivalent:

```
Function AsUShort(ByVal intVal As Integer) As Short
  AsUShort = intVal Mod 32768
    If intVal > 32767 Then
        AsUShort = AsUShort Or &H8000S
    End If
End Function
```

Applying the function yields the following results:

```
AsUShort(1) ==> 1
AsUShort(32767) ==> 32767
AsUShort(32768) ==> -32768
AsUShort(65535) ==> -1
```

A similar function is required to convert signed `Short` values from functions that return a `uint16` as a `Short`.

However, an alternative exists for passing constants into a function, which is to pass them as a hexadecimal literal suffixed with the `Short` character S e.g:

```
Const FOO As Short = &H8000S
```

Which Represents 32768 as a `Short`.

3.1.3 Execution Path

Irrespective of the programming language used, the application must be executed in a folder containing all of the required DLLs.

3.2 Using the TestEngine API

3.2.1 Building a C/C++ Application

To build a C or C++ application the `TestEngine.h` header file needs to be included in the source files. The application also needs to be linked to the `TestEngine.lib` import library. The file locations are:

- `TestEngine.h`: Include file `TestEngine.h` in**%TrueTestDir%\include**
- `TestEngine.lib`: Import library `TestEngine.lib` in**%TrueTestDir%\lib**

3.2.2 Building a Visual BasicApplication

To build a Visual Basic application add the TestEngine function declarations module to the project. There are modules for three versions of Visual Basic, these are:

- VB6 function declarations `TestEngineAPI.bas` in **%TrueTestDir%\include\VB6**
- VB7 (`.NET`) function declarations `TestEngineAPI.vb` in **%TrueTestDir%\include\VB7**
- VB8 (`2005 .NET`) and later function declarations `TestEngineAPI_05.vb` in **%TrueTestDir%\include\VB2005**

3.2.3 Building a Delphi Application

To build a Delphi application, add the `TestEngineAPI.pas` function declarations unit to the project. The unit file is in the following location:

- Delphi function declarations `TestEngineAPI.pas` in **%TrueTestDir%\include\Delphi**

3.2.4 Building a C# Application

To build a C# application, first add the `TestEngineAPI.cs` unit to the project. The unit file is in the following location:

- C# class wrapper file TestEngineAPI.cs in **%TrueTestDir%\include\C#**

Add the declaration using `TestEngineAPI;` to the top of any code files that call TestEngine functions. Call the static class member functions using the class name, e.g. `TestEngine.openTestEngine(…);`

3.2.5 Example Application

This section gives an example C++ program showing how to use the functions in the TestEngine API to write a value to the Persistent Store:

```
#include <iostream>
#include "testengine.h"

const uint16 PSKEY_USR0 = 650;
const uint16 MAX_KEY_SIZE = 20;

int main(int argc, char** argv)
{
    uint32 devHandle = openTestEngine(BCSP, "COM1", 115200, 5000, 0);

    if(devHandle != 0)
    {
        std::cout << "Device Handle = " << devHandle << std::endl;
        uint16 keySize;
        if(psSize(devHandle, PSKEY_USR0, PS_STORES_IFR, &keySize) == TE_OK)
        {
            std::cout << "Size of PSKEY_USR0 = " << keySize << std::endl;
        }
        else
        {
            std::cout << "psSize returned error" << std::endl;
            closeTestEngine(devHandle);
            return -1;
        }
        if(keySize == 0)
        {
            /* Doesn't exist, so need to create it */
            keySize = 1;
        }
        uint16 data[MAX_KEY_SIZE];
        for(uint16 i = 0; i < keySize; ++i)
        {
            data[i] = i;
        }
        if(psWrite(devHandle, PSKEY_USR0, PS_STORES_I, keySize, data) != TE_OK)
        {
            std::cout << "psWrite returned error" << std::endl;
            closeTestEngine(devHandle);
            return -1;
        }
        closeTestEngine(devHandle);
    }
    else
    {
        std::cout << "openTestEngine returned invalid handle" << std::endl;
        return -1;
    }

    return 0;
}
```

3.3 Using the TestFlash API

There are currently two flash programming APIs supported by TestFlash, the legacy API, and the newer, multiple connection API. The multiple connection API allows multiple device connections per process, e.g. to multiple BlueCore devices using multiple USB-SPI interfaces. The functions in one API should not be interleaved with those from the other API. See the TestFlash help file for more information.

3.3.1 Building a C/C++ Application

To build a C or C++ application you need to include the `TestFlash.h` and/or `TestPS.h` header files in the source files. The application also must be linked against the import libraries. These files are in the following locations:

- Include files `TestFlash.h` and `TestPS.h` in **%TrueTestDir%\include**
- Import library `TestFlash.lib` in **%TrueTestDir%\lib**

3.3.2 Building a Visual Basic Application

To build a Visual Basic application add the TestFlash function declarations module to the project. There are modules for three versions of Visual Basic. The files are in the following locations:

- VB6 function declarations `TestFlashAPI.bas` in **%TrueTestDir%\include\VB6**
- VB7 (`.NET`) function declarations `TestFlashAPI.vb` in **%TrueTestDir%\include\VB7**
- VB8 (`2005 .NET`) and later function declarations `TestFlashAPI_05.vb` in **%TrueTestDir%\include\VB2005**

3.3.3 Building a Delphi Application

To build a Delphi application, add the `TestFlashAPI.pas` function declarations unit to the project. The unit file is in the following location:

- Delphi function declarations `TestFlashAPI.pas` in **%TrueTestDir%\include\Delphi**

3.3.4 Building a C# Application

To build a C# application, first add the `TestFlashAPI.cs` unit to the project. The unit file is in the following location:

- C# class wrapper file TestFlashAPI.cs in **%TrueTestDir%\include\C#**

Add the declaration using `TestFlashAPI;` to the top of any code files that call TestEngine functions. Call the static class member functions using the class name, e.g. `TestFlash.fileInit(…);`

3.3.5 Example Application

This section gives a sample C++ program showing how to use the functions in the TestFlash legacy API to program Flash (non-blocking method for a single device):

```
#include <iostream>
#include <windows.h>
#include "testflash.h"
int main(int argc, char** argv)
{
    if(flInit(1, 26, 1, TFL_SPI_LPT) != TFL_OK)
    {
        std::cout << "Failed to initialise flash" << std::endl;
        return -1;
    }
    if(flReadProgramFiles("testBuild\\default") != TFL_OK)
    {
        std::cout << "Failed to read flash program files" << std::endl;
        flClose();
        return -1;
    }
    if(flProgramSpawn() != TFL_OK)
    {
        std::cout << "Failed to spawn flash program thread" << std::endl;
        flClose();
        return -1;
    }
    int32 progress = flGetProgress();
    while(progress < 100)
    {
        std::cout << "Programming..." << progress << "%" << std::endl;
        Sleep(1000);
        progress = flGetProgress();
    }
    std::cout << "Completed" << std::endl;
    int32 error = flGetLastError();
    if(error != TFL_OK)
    {
        std::cout << "Programming failed with error: " << error <<
        std::endl;
        flClose();
        return -1;
    }
    std::cout << "Successfully programmed device" << std::endl;
    flClose();
    return 0;
}
```

3.4 Using the TestE2 API

3.4.1 Building a C/C++ Application

To build a C or C++ application include the `TestE2.h` header file in the source files. The application also must be linked against the import libraries. These files are in the following locations:

- Include file `TestE2.h` in **%TrueTestDir%\include**
- Import library `TestE2.lib` in **%TrueTestDir%\lib**

3.4.2 Building a Visual Basic Application

To build a Visual Basic application add the TestE2 function declarations module to the project. There are modules for three versions of Visual Basic. The files are in the following locations:

- VB6 function declarations `TestE2PI.bas` in **%TrueTestDir%\include\VB6**
- VB7 (`.NET`) function declarations `TestFlashAPI.vb` in **%TrueTestDir%\include\VB7**
- VB8 (`2005 .NET`) and later function declarations `TestE2API_05.vb` in **%TrueTestDir%\include\VB2005**

3.4.3 Building a Delphi Application

To build a Delphi application, add the `TestE2API.pas` function declarations unit to the project. The unit file is in the following location:

- Delphi function declarations `TestE2API.pas` in **%TrueTestDir%\include\Delphi**

3.4.4 Building a C# Application

To build a C# application add the `TestE2API.cs` unit to the project. The unit file is in the following location:

- C# class wrapper file `TestE2API.cs` in **%TrueTestDir%\include\C#**

Add the declaration using `TestE2API;` to the top of any code files that will call TestE2 functions. Call the static class member functions using the class name, e.g. `TestE2.te2Open(…);`.

3.4.5 Example Application

This section gives an example C++ program showing how to use the functions in the TestE2 API to program and verify the contents of an EEPROM in non-blocking mode. The device being accessed is attached to port LPT1:

```
#include "TestE2.h"
#include <iostream>
#include <windows.h>

using namespace std;

int main(int argc, char* argv[])
{
    if(te2Open(TE2_SPI_LPT, 1, 0) != TE2_OK)
    {
        cout << "Error opening connection: " << te2GetLastError() << endl;
        return -1;
    }

    // Program the device from an image file, don't wait for completion
    if(te2Program("eeImage.xpv", 0) != TE2_OK)
    {
        cout << "Error programming: " << te2GetLastError() << endl;
        te2Close();
        return -1;
    }
    // Wait for programming completion
    cout << "Programming";
    while(te2IsBusy() != 0)
    {
        cout << '.';
        Sleep(100);
    }
    cout << endl;
    if(te2GetResult() != TE2_OK)
    {
        cout << "Error programming: " << te2GetLastError() << endl;
        te2Close();
        return -1;
    }

    // Verify the contents
    if(te2Verify("eeImage.xpv", 0) != TE2_OK)
    {
        cout << "Verify failed: " << te2GetLastError() << endl;
        te2Close();
        return -1;
    }
    // Wait for verify completion
    cout << "Verifying";
    while(te2IsBusy() != 0)
    {
        cout << '.';
        Sleep(100);
    }
    cout << endl;
    if(te2GetResult() != TE2_OK)
    {
        cout << "Error verifying: " << te2GetLastError() << endl;
        te2Close();
        return -1;
    }

    cout << "Device programming successful" << endl;
    te2Close();
    return 0;
}
```

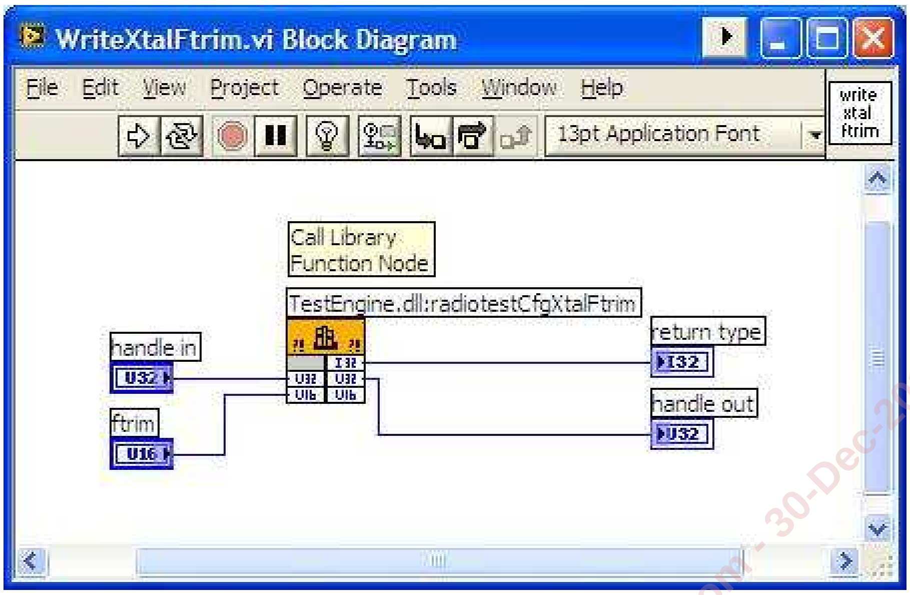
3.5 Using LabView

3.5.1 Introduction

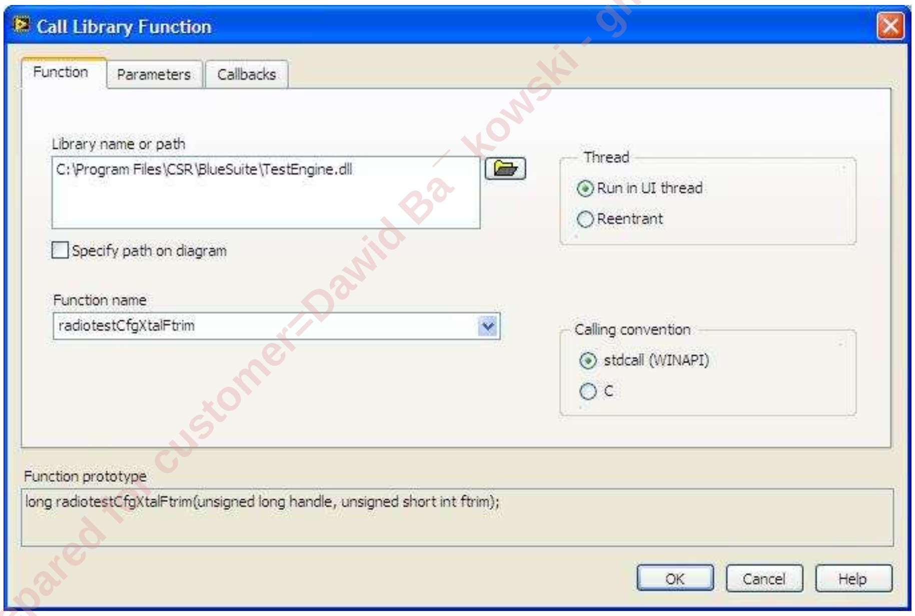
This section describes calling the TestEngine DLL functions from the LabView Development Environment. The examples use LabView Professional Development System 8.2 but the same principles apply for versions 8.0, 7.1, 7.0 and 6.0. The same methods can be used when using TestFlash or TestE2 with LabView.

3.5.2 Basic Function Calling

Use the LabView Call Library Function Node to call the TestEngine DLL contained in **c:\Program Files\CSR\Bluesuite**. A handle to a device is obtained by calling one of the `openTestEngine`… functions.


WriteXtalFtrim.vi Block Diagram
File Edit View Project Operate Tools Window Help
13pt Application Font
write xtal ftrim
Call Library Function Node
TestEngine.dll:radiotestCfgXtalFtrim
handle in
U32
ftrim
U16
return type
I32
handle out
U32

Right-click on the **Call Library Function Node** and configure the **Function** tab as shown. Select the function name from the drop-down list, making sure the calling convention is set to **stdcall(WINAPI)**.


Call Library Function

Function | Parameters | Callbacks

Library name or path: C:\Program Files\CSR\BlueSuite\TestEngine.dll

☐ Specify path on diagram

Function name: radiotestCfgXtalFtrim

Thread: [✓ Run in UI thread] Reentrant

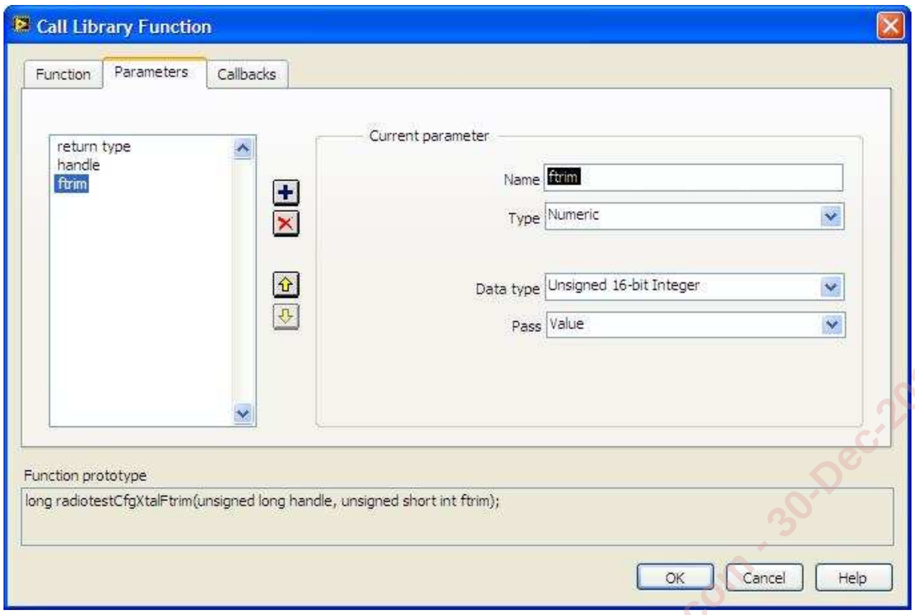
Calling convention: [✓ stdcall (WINAPI)] C

Function prototype: long radiotestCfgXtalFtrim(unsigned long handle, unsigned short int ftrim);

OK Cancel Help

On the **Parameters** tab, add parameters to match those defined in **c:\Program Files\CSR\BlueSuite\include\TestEngine.h**:

```
    TESTENGINE_API(int) radiotestCfgXtalFtrim(uint32 handle, uint16 ftrim);
```



return type
handle
ftrim

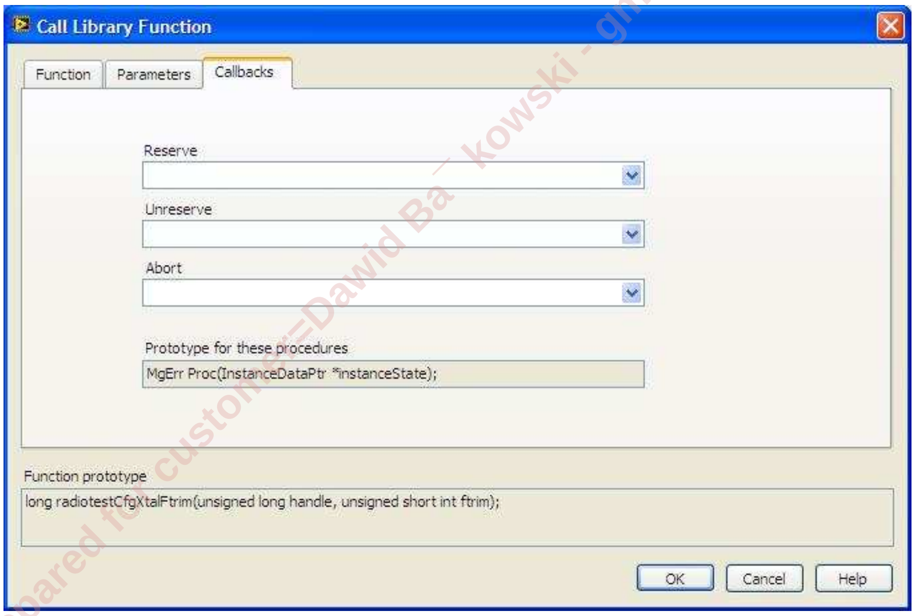
Current parameter

Name: ftrim
Type: Numeric
Data type: Unsigned 16-bit Integer
Pass: Value

Function prototype

long radiotestCfgXtalFtrim(unsigned long handle, unsigned short int ftrim);

Ensure that no callbacks are set for the function:



Reserve:

Unreserve:

Abort:

Prototype for these procedures

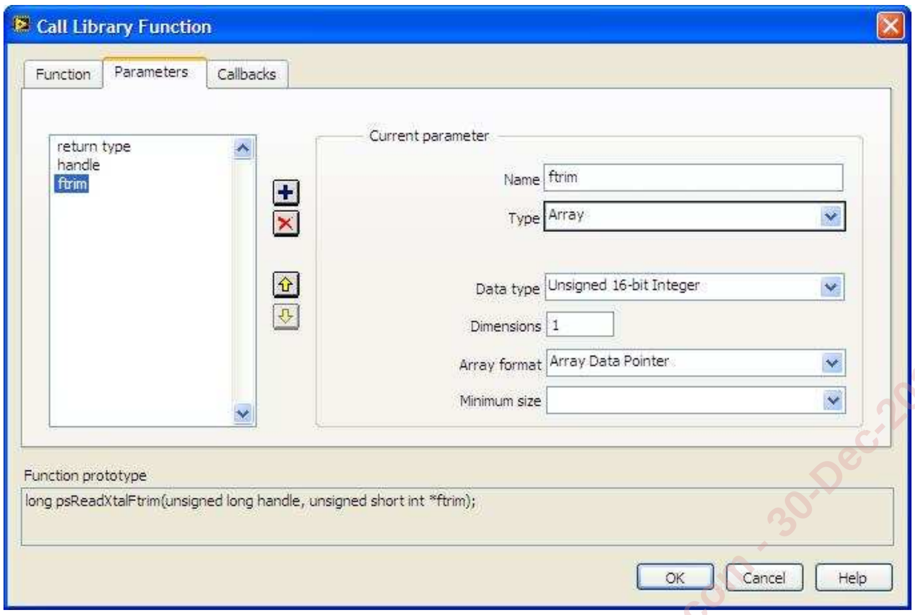
MgErr Proc(InstanceDataPtr *instanceState);

Function prototype

long radiotestCfgXtalFtrim(unsigned long handle, unsigned short int ftrim);

3.5.3 Reading Values Using Pointers

When reading values using pointers, use an array as the data structure e.g. for `psReadXtalFtrim`:


Call Library Function

Function | Parameters | Callbacks

return type
handle
ftrim

Current parameter

Name: ftrim

Type: Array

Data type: Unsigned 16-bit Integer

Dimensions: 1

Array format: Array Data Pointer

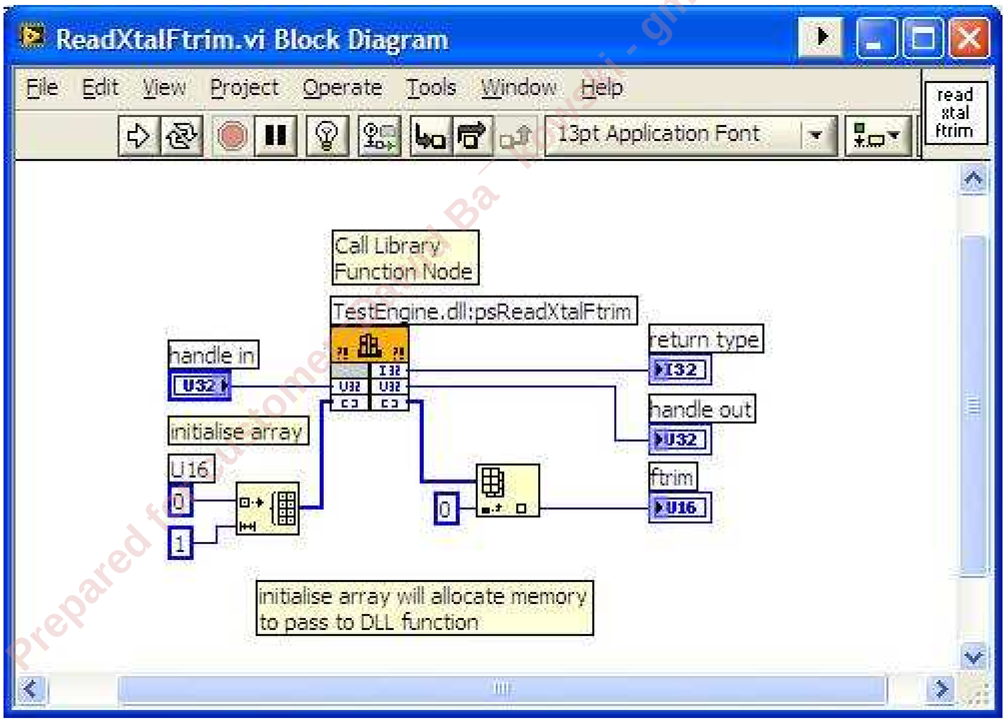
Minimum size:

Function prototype

```
long psReadXtalFtrim(unsigned long handle, unsigned short int *ftrim);
```

OK | Cancel | Help

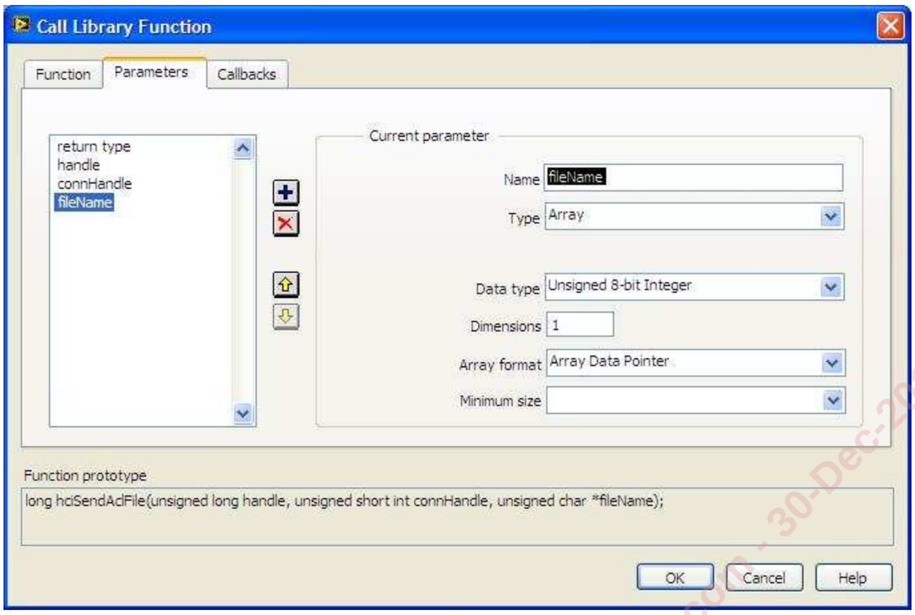
Use the initialise array function to allocate memory before calling the DLL function.


ReadXtalFtrim.vi Block Diagram

File Edit View Project Operate Tools Window Help
13pt Application Font
read xtal ftrim
Call Library Function Node
TestEngine.dll:psReadXtalFtrim
handle in
U32
return type
I32
handle out
U32
initialise array
U16
0
1
0
ftrim
U16
initialise array will allocate memory to pass to DLL function

3.5.4 Passing a String to a LabView DLL Function

To pass a string to a DLL function, use an array of unsigned chars:



return type
handle
connHandle
fileName

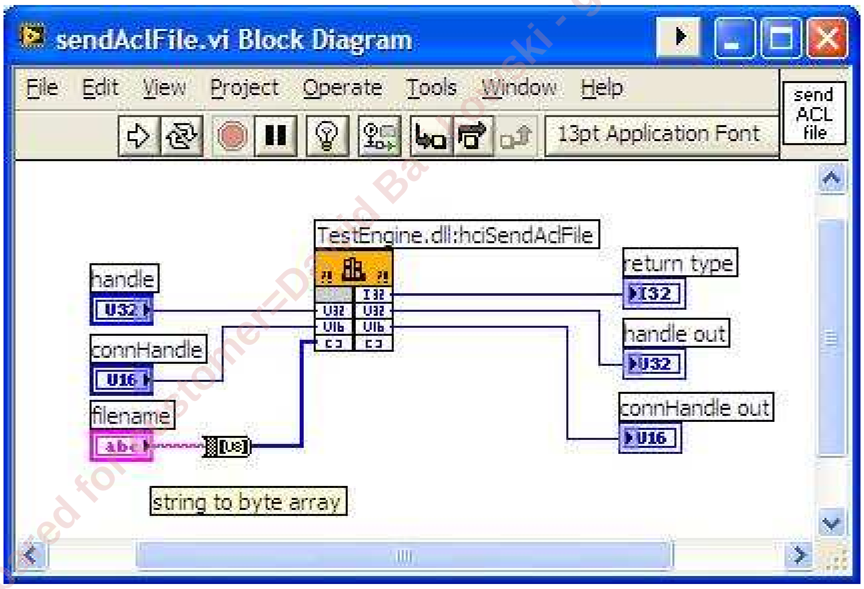
Current parameter

Name: fileName
Type: Array
Data type: Unsigned 8-bit Integer
Dimensions: 1
Array format: Array Data Pointer
Minimum size:

Function prototype

long hciSendAclFile(unsigned long handle, unsigned short int connHandle, unsigned char *fileName);

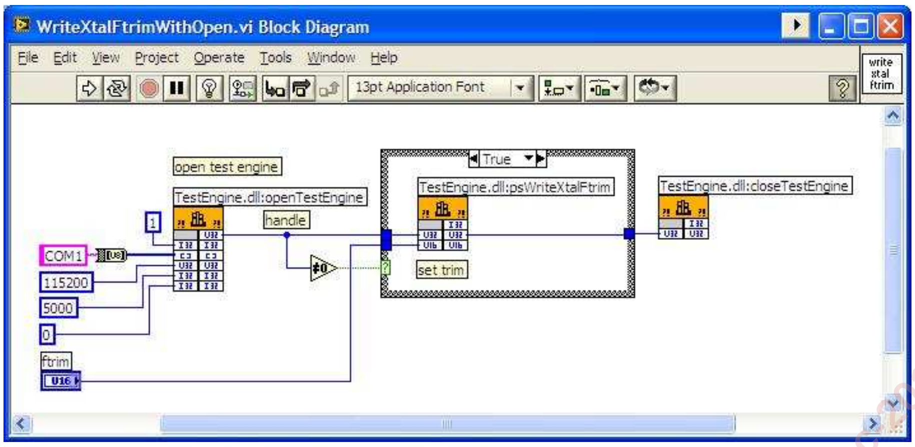
Use the string to byte array function to convert a LabView string to a byte array before passing this to the TestEngine DLL.



3.5.5 OpenTestEngine, OpenTestEngineSpi and CloseTestEngine

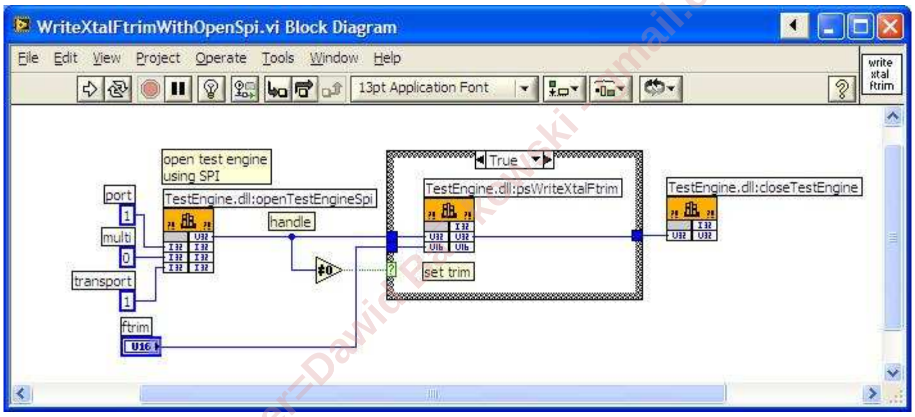
To obtain a handle to a device, call `openTestEngine`. Wire the output of `openTestEngine` to any function that you want to call. This handle should be passed through all functions that are called while testing.

Ensure that `closeTestEngine` is called after testing is completed.


WriteXtalFtrimWithOpen.vi Block Diagram
File Edit View Project Operate Tools Window Help
13pt Application Font
open test engine
TestEngine.dll:openTestEngine
handle
True
TestEngine.dll:psWriteXtalFtrim
set trim
TestEngine.dll:closeTestEngine
COM1
115200
5000
0
ftrim
U16

To communicate with a device using the SPI interface, call `openTestEngineSpi` and wire the output to the function that you want to call. Some TestEngine functions may not be available when using `openTestEngineSpi`, further information is contained in the TestEngine help files.

Ensure that `closeTestEngine` is called when any operation(s) are complete.


WriteXtalFtrimWithOpenSpi.vi Block Diagram
File Edit View Project Operate Tools Window Help
13pt Application Font
open test engine using SPI
TestEngine.dll:openTestEngineSpi
handle
True
TestEngine.dll:psWriteXtalFtrim
set trim
TestEngine.dll:closeTestEngine
port
multi
transport
ftrim
U16

4 Terms and Definitions

API	Application Programming Interface
BCCMD	BlueCore Command
BCSP	BlueCore Serial Protocol
BlueCore®	Group term for CSR's range of Bluetooth wireless technology ICs
BlueSuite®	Suite of development tools that contains BlueCore utilities
BlueTest3	Software for executing BlueCore's Built-in Self-test functions
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
BTCli	Bluetooth Command Line Interface
COFF	Common Object File Format
CSR	Cambridge Silicon Radio
DLL	Dynamically Linked Library
DM	Device Manager
e.g.	*exempli gratia*, for example
EEPROM	Electrically Erasable Programmable Read Only Memory
etc.	*et cetera*, and the rest, and so forth
H4	UART-based HCI transport, described in section of H4 of v1.0b of Bluetooth Specification
HCI	Host Controller Interface
HQCMD	Host Query Command
i.e.	*id est*, that is
IC	Integrated Circuit
OMF	Object Module Format
Persistent Store (PS)	Storage of BlueCore's configuration values in non-volatile memory
RFCOMM	Protocol layer providing serial port emulation over L2CAP; element of Bluetooth
SPI	Serial Peripheral Interface
TrueTest®	A set of libraries for developing production test applications for BlueCore-enabled devices
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

VB	Microsoft® Visual Basic®