# LabVIEW

Software Engineering Technical Manual and Exercises

Version 4.1, August 2013

NATIONAL INSTRUMENTS™

## CONTENTS

**To download a copy of this manual and the latest version of LabVIEW code referenced in the exercises, please visit: http://bit.ly/lv_swe**

## INTRODUCTION TO SOFTWARE ENGINEERING

LabVIEW is a graphical system design environment containing all of the tools that engineers and scientists need to build some of today's most technologically challenging and advanced systems. As the complexity of LabVIEW applications has grown, it's become increasingly important that a structured and disciplined development approach be applied in order to deliver high-quality, professional applications.

For these applications, a very structured and regimented programming process must be followed to ensure quality and reliability of the overall system. This guide will examine the development life-cycle and explain some of the tools that can improve and automate common software engineering practices.

## SOFTWARE CONFIGURATION MANAGEMENT

Many developers have experienced the frustration of unmanaged environments, where people overwrite each other's changes or are unable to track revisions. Managing a large number of files or multiple developers is a challenge in any language. In fact, it's often a challenge to manage an application even if it's just one developer working on a small to medium application. Large development projects rely upon configuration management tools to satisfy the following goals:

1. Define a central repository of code
2. Manage multiple developers
3. Detection and resolution of code collisions
4. Tracking behavioral changes
5. Identification of changes and who made them
6. Ensuring everyone has latest copy of code
7. Backing up old code versions
8. Managing *all* files, not just source code

Perhaps the most important and commonly known SCM tool is source code control (SCC). However, in addition to many third party SCC tools, we'll see that there are a number of additional tools available in the LabVIEW development environment that is designed to help with these goals.

Establishing guidelines for storing and managing files requires foresight into how the application will be structured, how functionality will be divided, and the types of files beyond source code that will be important to keep track of. Devote time to making decisions about how functionality will be divided among code and to working with developers on file storage locations and the additional files or resources they will need to function properly.

Source code control provides an interface for submitting and retrieving changes from a centralized repository where the code is stored. This repository should be setup in such a way that all team members have regular access in order to retrieve updates from other team members and submit their work. In addition to simplifying access to the code, the repository contains all submitted revisions of source code, making it possible to revert to a previous version and compare changes that have been introduced over-time. This is especially valuable in situations where changes introduce unexpected or undesirable behavior that requires identifying the root cause.

A centralized repository also ensures that conflicting changes can be detected and addressed. These conflicts typically arise when two or more people make modifications to the same source code or file. Source code control is designed to detect and flag these problems, and can step through merging these changes.

# Exercise 1: Creating a SVN Repository and Adding Code

## Goal

This exercise will explain the most fundamentally important concepts required to use source code control with any LabVIEW system.

## Scenario

We want to store our source code in a subversion repository in order to track revisions, identify changes, and share code.

## Description

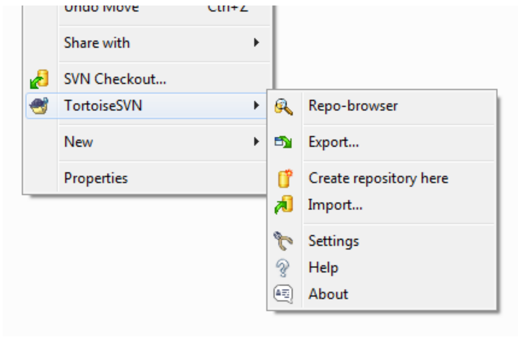
We are going to create a new subversion repository, add our LabVIEW project and then check the code out in order to begin programming.

## Concepts Covered

- Creating a subversion repository
- Adding code

## Setup (Complete the following in order listed)

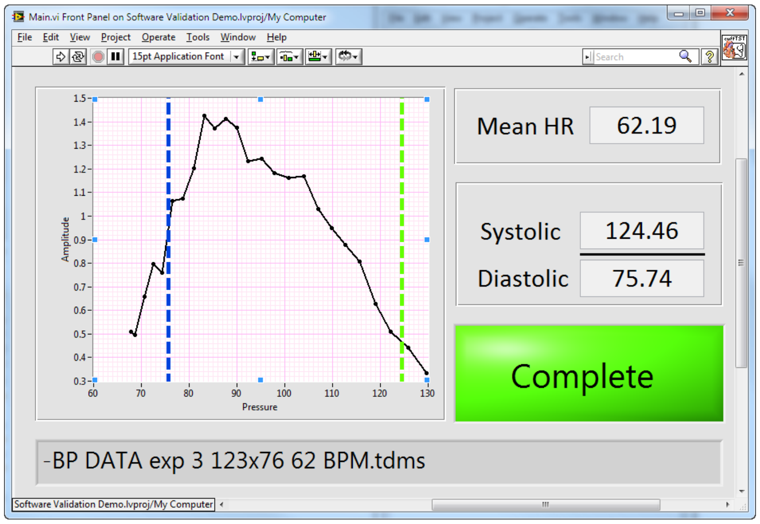
- Ensure that LabVIEW 2013 (or later) is installed
- Ensure TortoiseSVN is installed
  - If installed, the right-click menu in Windows Explorer will have 'TortoiseSVN' listed with a sub-menu of available functionality. Verify that you see a similar display when you right-click anywhere inside of Windows Explorer.



  - If not present, download and install from the web. Be sure to restart your computer aftewrads
- Ensure Viewpoint's Free TortoiseSVN Toolkit is installed from the LabVIEW Tools Network, visit ni.com/labviewtoolsnetwork **(you will need to restart LabVIEW after installing this product)**

## Instructions

1. Open a clean copy of the project's source code that will be added to the source code control repository.
    a. From the LabVIEW getting started screen, click 'Create Project.' Filter by 'Demonstrations' on the left of this dialog, select 'Software Engineering Tools' and click **Next**.
    b. Select a temporary location on disk to create this project, such as '…Desktop\Software Engineering Tools'
2. Quickly familiarize yourself with what the system does
    a. Once the project is created, open the project file and open the front panel of main.vi. This application simulates a device that computes blood pressure based upon input from a pressure transducer. Instead of connecting hardware, this version will compute the ratio of diastolic to systolic pressure based upon recorded patient data. In order to compute this value, the application uses a very simple state machine that includes some basic acquisition (in this case, from a file), simple filtering and signal processing. Click the run arrow to start the application
    b. Since we do not have actual hardware, you will be prompted for previously recorded data. Select, 'BP DATA exp 3 123x76 62 BPM'
    c. Click 'Take Blood Pressure' to begin acquiring the patient data (it will simulate the timing of the acquisition – to skip ahead, disable the 'Timing' Boolean. *Note: this control is intentionally mis-spelled, as we will detect this in a later analysis step.*
    d. The final screen should display the results of the analysis, as shown below:


Main.vi Front Panel on Software Validation Demo.lvproj/My Computer

Mean HR 62.19

Systolic 124.46

Diastolic 75.74

Complete

-BP DATA exp 3 123x76 62 BPM.tdms

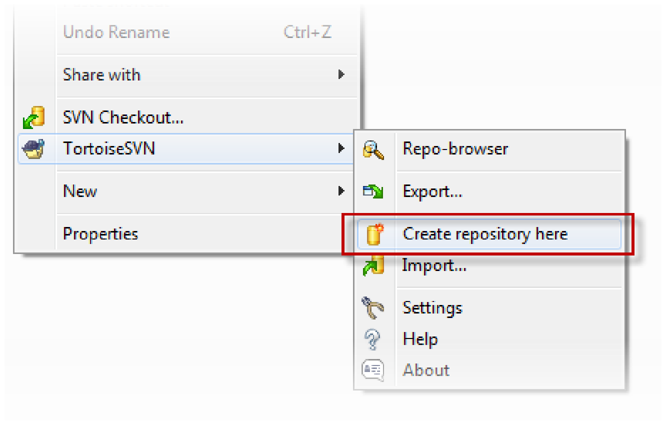
File Edit View Project Operate Tools Window Help
15pt Application Font
Amplitude
Pressure
Software Validation Demo.lvproj/My Computer

    a. **Close the project** – this will be the source that we commit to the repository in the next few steps. If you made any changes, *do not save them*.

3. Create a local Subversion repository (this is only necessary if you do not already have a repository that you plan to use). For the sake of simplicity, this exercise recommends placing the repository in the root directory: C:\. **Note**: if using a Subversion repository for actual development, it is strongly recommended that the repository be created and stored on a remote server that all team members have a connection to.

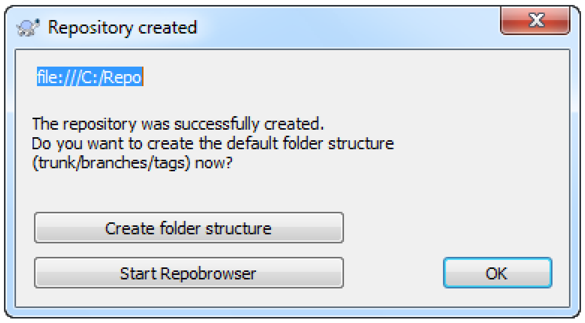
a. Navigate to the root directory of your computer and create a new folder called 'Repo.' When created, the path should be C:\Repo

b. Open this directory and expand the TortoiseSVN (right-click in the explorer window) and select 'Create Repository Here.' This operation creates the database where your source will be stored and managed.



c. Open the repository browser to view this repository in the TortoiseSVN interface

   i. **Option 1:** Newer versions of TortoiseSVN may display a prompt to open the repository browser after completing the previous step. If prompted, select 'Start Repobrowser' (as shown below)

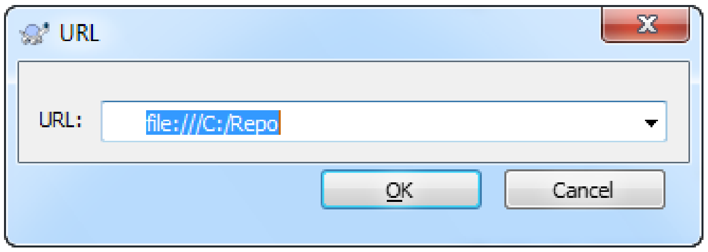


Repository created

file:///C:/Repo

The repository was successfully created.
Do you want to create the default folder structure
(trunk/branches/tags) now?

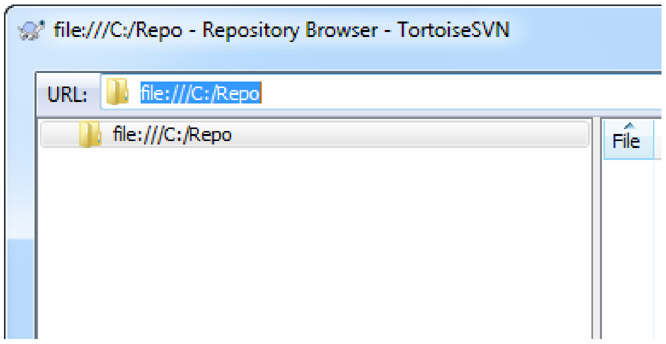
   ii. **Option 2 (for older versions of Tortoise):** If you do not receive a prompt, open the repository browser from the TortoiseSVN right-click menu by selecting 'TortoiseSVN> Open repo-browser.' When prompted, enter file:///C:/Repo and press 'OK'



URL

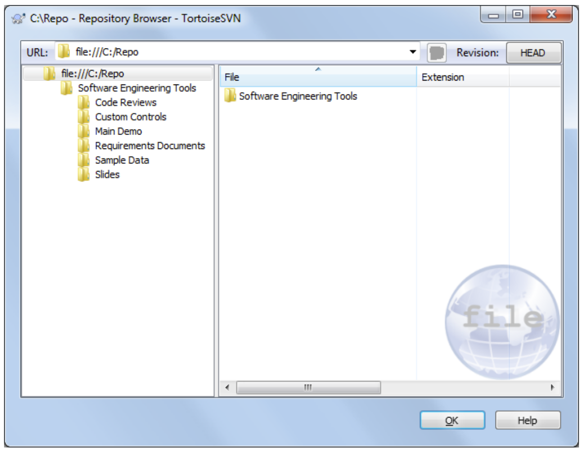
URL: file:///C:/Repo

d. The repository browser should now be displaying the contents of the newly created repository, which will be empty (as shown below). This is where we will add our files in the next step.



**NOTE:** You will never directly modify the contents of 'C:\Repo' on disk (ie: using Windows Explorer). This is the subversion database – you will use the TortoiseSVN client to send and receive code to the database.

4. Add your LabVIEW application to the subversion repository (if you skipped the previous step, make sure that you have opened the TortoiseSVN repository browser and pointed it to the appropriate location)
   a. Right-click in the right pane of the repository browser and select 'Add folder…' to import the source code
   b. Select the path of the source code to be added ('…Desktop\Software Engineering Tools') if you followed the recommendation in the first step), and click 'Select Folder.' When prompted to enter a log message, type 'adding files to the repository' and press 'OK' Tortoise will think for a few seconds while these files are imported to the database.
   c. Verify that the repo-browser now contains the source files by double clicking on the newly added folder. The browser should now display the contents of the project that are stored in the subversion repository.



   d. Return to the original path on disk '…Desktop\Software Engineering Tools.' Since we've added the files on disk to the repository, you can safely delete the folder on disk and its contents, as a copy of these files now resides in the Subversion repository.

# Exercise 2: Checking out code and committing changes

## Goal

This exercise will familiarize users who are new to SCC with the practice of checking out code and committing regular updates

## Scenario

A subversion repository has been created and contains the code we need. We will check out a copy locally, begin development and commit changes as they are made.

## Description

When using Subversion, to 'checkout code' refers to downloading a copy of the code to your local development machine. Unlike other tools, you do not have to checkout the code every time you intend to make a change. Once checked out, developers have all of the files locally. If they make changes, the file will have a red icon overlay until that change is committed to the repository. If another developer needs these latest changes, they'll simply invoke the 'Update' operation on the file or containing folder.

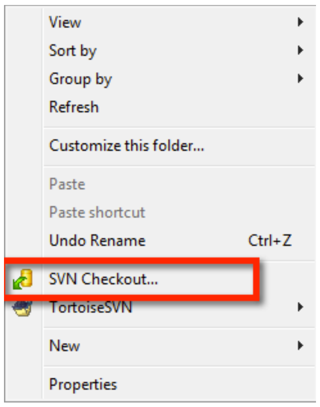
Note that some development teams will opt to lock files to prevent multiple people from modifying the same file at once. This behavior is not covered.

## Concepts Covered

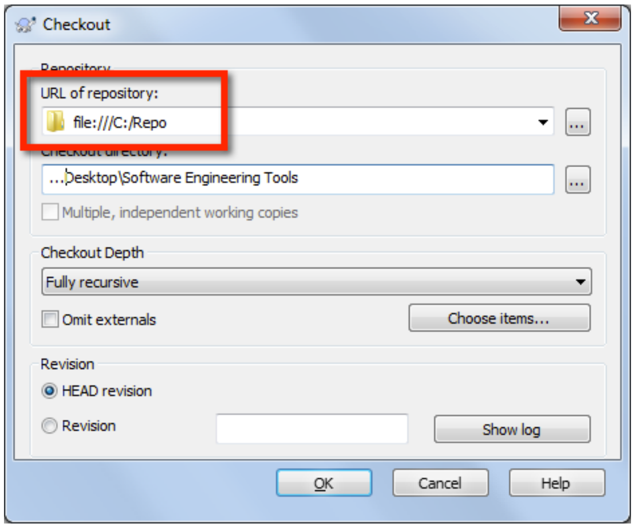
- Checking out code
- Committing changes
- Viewing a log of revisions
- Reverting changes

## INSTRUCTIONS

1. Checkout a version controlled instance of the application
   a. Use Windows Explorer to navigate to the folder location where you would like to checkout an instance of this application - this location will be where development is done once the code has been checked out. It is recommended that you create a new folder on the Desktop labeled 'Software Engineering Tools.'
   b. Inside the folder, right click to bring up the TortoiseSVN menu and select **SVN Checkout…**



   c. If you used the directory locations and file names in Exercise 1, the path to the repository will be file:///C:/Repo



URL of repository: file:///C:/Repo

Checkout directory: ...Desktop\Software Engineering Tools

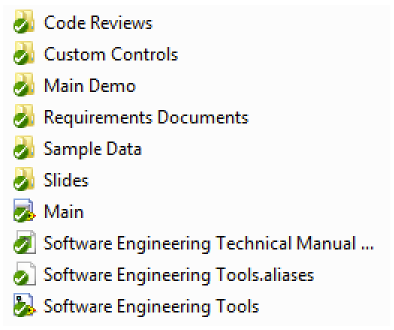
☐ Multiple, independent working copies

Checkout Depth: Fully recursive

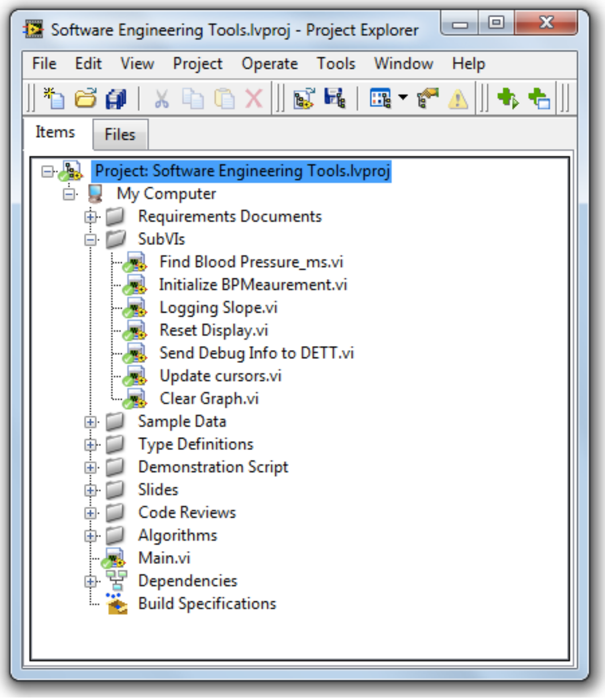
☐ Omit externals

Revision: ● HEAD revision ○ Revision

   d. The 'Checkout directory' should be set to the location where you clicked. Make sure that the URL of repository is set to the correct location – if using defaults, it will be file:///C:/Repo
   e. Press **OK**. Tortoise will download a local copy of the source files that will be version managed by subversion.
   f. Once checkout has finished, verify that the files have been properly checked out and are being tracked by subversion. If this operation is completed successfully, the files will have status icon overlays in Windows Explorer, which should be all green (as shown).

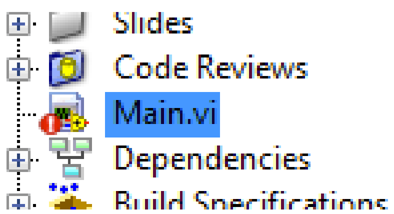

Code Reviews
Custom Controls
Main Demo
Requirements Documents
Sample Data
Slides
Main
Software Engineering Technical Manual ...
Software Engineering Tools.aliases
Software Engineering Tools

g. Open the application in LabVIEW by double-clicking on 'Software Engineering Tools.lvproj'
h. If Viewpoint's TortoiseSVN Plugin is installed, the icons will also be overlaid in the Project Explorer (note that they do not appear on virtual folders, but will be overlaid on their contents).


Software Engineering Tools.lvproj - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files

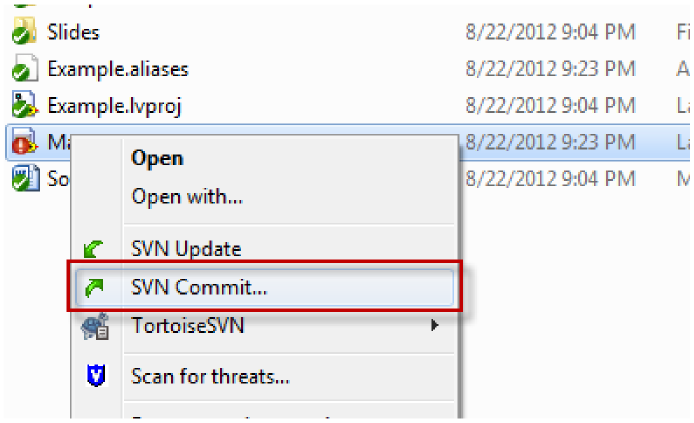
```
Project: Software Engineering Tools.lvproj
  My Computer
    Requirements Documents
    SubVIs
      Find Blood Pressure_ms.vi
      Initialize BPMeaurement.vi
      Logging Slope.vi
      Reset Display.vi
      Send Debug Info to DETT.vi
      Update cursors.vi
      Clear Graph.vi
    Sample Data
    Type Definitions
    Demonstration Script
    Slides
    Code Reviews
    Algorithms
    Main.vi
    Dependencies
    Build Specifications
```

2. Practices making changes and committing them committing them to the repository
   a. Open Main.vi from the Project Explorer
   b. Press **CTRL+E** to switch to the block diagram
   c. Make changes to the block diagram – feel free to do and/or change anything, as changes will be undone in a later step. Some recommendations:
      i. Move items on block diagram
      ii. Add cases to case structure(s)
      iii. Delete wires from block diagram
   d. Save the VI (Press **CTRL+S**)
   e. The TortoiseSVN icon overlay should now be red, indicating that this file has been modified, but not yet committed to the repository.



f. Right-click on 'Main.vi' on disk and select **SVN Commit…** to send the change to the repository. This can be done from within LabVIEW (if using the Viewpoint TortoiseSVN plugin), or from Windows Explorer.

**Note:** You can also invoke a commit operation from Quick Drop during development. With the VI open, launch Quick Drop by pressing **CTRL+Space** and press **CTRL+C** to commit.

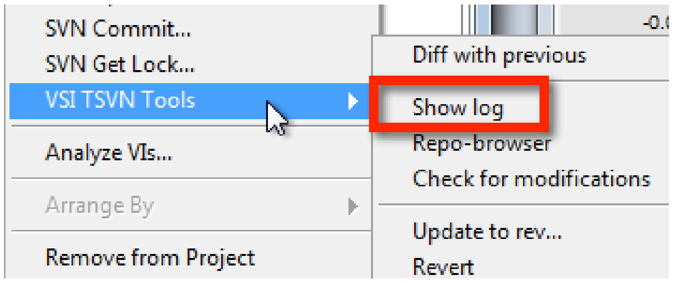


g. When prompted to enter a log message, write a brief description of the change you're making, such as 'changing the initial state.' Press **OK** when done and this modification will be submitted.

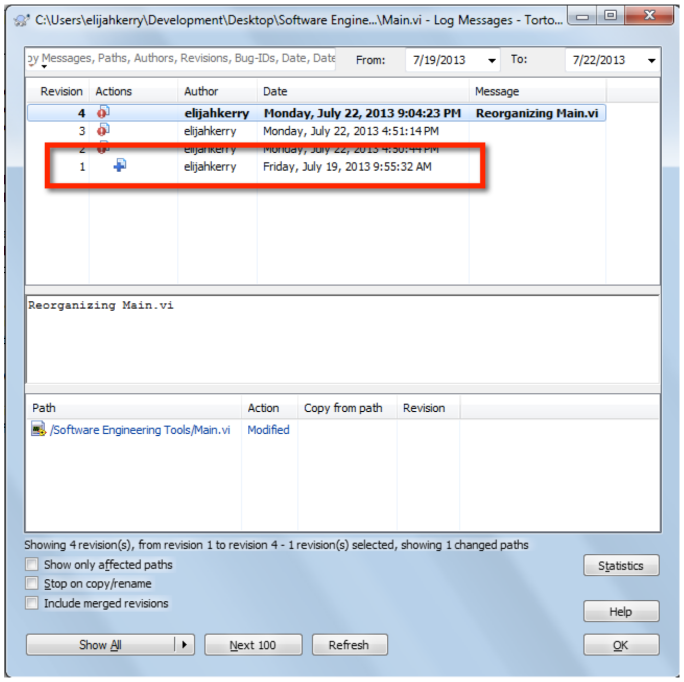
h. Feel free to repeat the instructions in step 2 multiple times in order to practice committing multiple changes.

**Note:** Changes that have been committed can be downloaded by other developers who are connected to the same repository. The other developers will just need to select the **SVN Update** menu item for the files or folders they want to update. To simulate this, you can repeat the steps in this exercise to checkout another local copy from the same repository.

3. View the history of revisions you've made by using the Subversion log viewer
   a. Right-click on Main.vi in the Project Explorer and click **Show log**. This option will appear under VSI TSV Tools, as shown below:



b. The log showing all of the changes created in step 2 should be visible:


C:\Users\elijahkerry\Development\Desktop\Software Engine...\Main.vi - Log Messages - Torto...

From: 7/19/2013 To: 7/22/2013

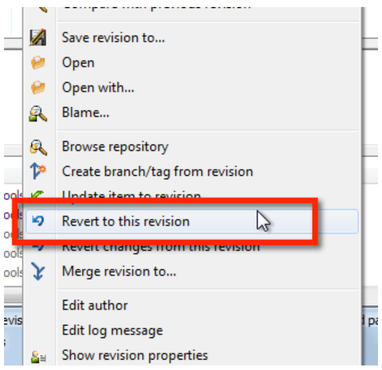
| Revision | Actions | Author | Date | Message |
|---|---|---|---|---|
| 4 | | elijahkerry | Monday, July 22, 2013 9:04:23 PM | Reorganizing Main.vi |
| 3 | | elijahkerry | Monday, July 22, 2013 4:51:14 PM | |
| 2 | | elijahkerry | Monday, July 22, 2013 4:50:44 PM | |
| 1 | | elijahkerry | Friday, July 19, 2013 9:55:32 AM | |

```
Reorganizing Main.vi
```

| Path | Action | Copy from path | Revision |
|---|---|---|---|
| /Software Engineering Tools/Main.vi | Modified | | |

Showing 4 revision(s), from revision 1 to revision 4 - 1 revision(s) selected, showing 1 changed paths

4. Revert to the unmodified version we started with
   a. Make sure that the front panel of main.vi has been closed
   b. Revision 1 represents the original copy of Main.vi that was committed to the repository. Right-click on revision 1 and select **Revert to this revision** from the menu that appears.



   c. Reopen main.vi and observe that it has reverted to the original copy of code
5. Commit the reverted VI
   a. Right-click on Main.vi and commit it to the repository. This will ensure that the original, working VI (which will be needed for later exercises) is now the most recent version that's been submitted

# EXERCISE 3: TRACKING CHANGES TO VIS USING SCC

## GOAL

We want to be able to compare two copies of a VI to identify what has changed through the use of source code control.

## SCENARIO

We are developing a LabVIEW application with the help of a team of developers. In preparation for a code review, we want to compare our most recent changes with the previous version.

## DESCRIPTION

We are going to download the most recent code and be able to compare changes we make with previous versions using the graphical differencing feature of LabVIEW. After making undesirable changes and saving them, we will be able to revert to a previous version.

## CONCEPTS COVERED

- Tracking changes with source code control
- Graphical differencing from outside the development environment
- Reverting to a previous version

## SETUP

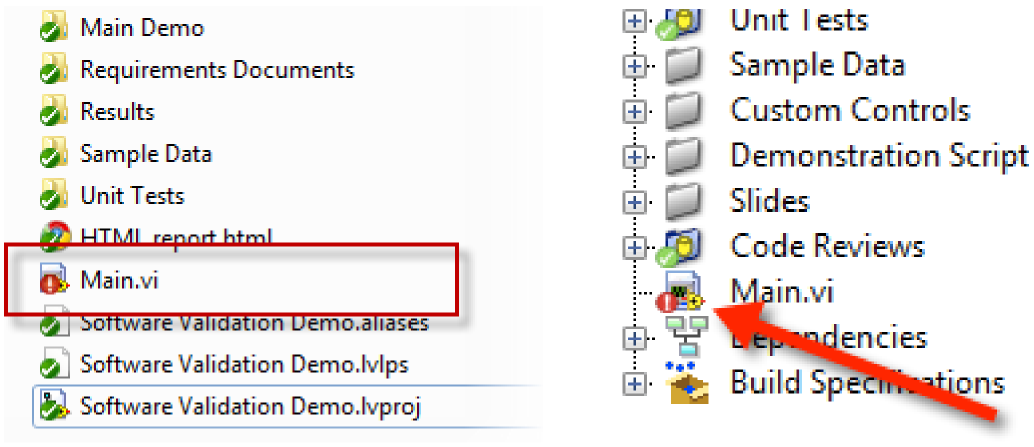
If you choose to not use Viewpoint's tool in Exercise 1, you can manually configure the Tortoise client to invoke diff and merge for VIs using the following steps (the toolkit referenced above does this for you in addition to adding shortcuts inside of LabVIEW):

- Right click in Windows Explorer
- Select **TortoiseSVN > Settings > Diff Viewer**
- Select **Advanced** and enter the following for a .vi file type (this can also be used for a .ctl). Do this by clicking 'Add' in the **Advanced** dialog

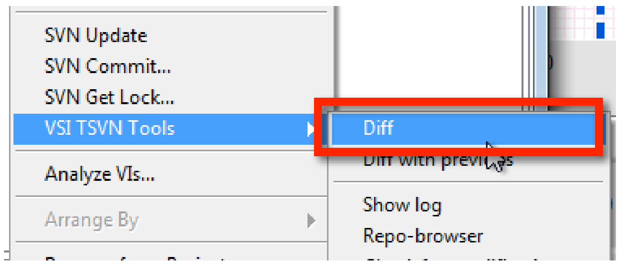
*"C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe" %mine %base -nobdcosm -nobdpos*

## INSTRUCTIONS

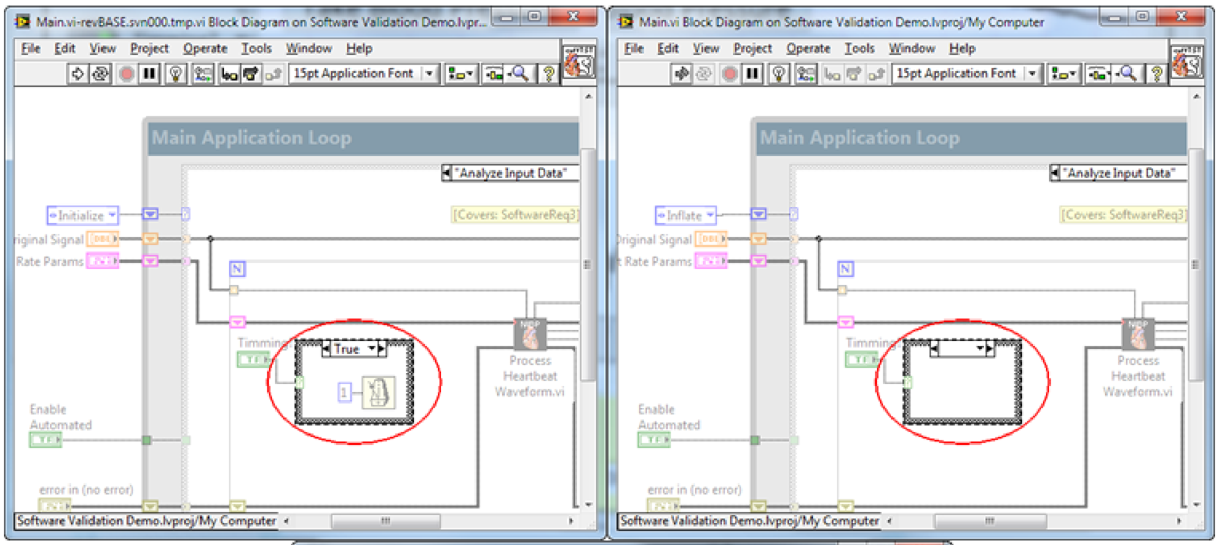
1. Make changes and compare them with the previous version.
    a. Switch to the block diagram of Main.vi.
    b. Make several changes that could introduce bugs, unexpected behavior or even break execution. Suggested modifications:
        1. Change timing parameters (especially hard to find, but can cause significant problems)
        2. Add cases to case structures
        3. Change block diagram constants
        4. Delete and/or move code
    b. Save the modifications
    c. Observe that **Main.vi**, which has been modified, has a red exclamation mark over the icon of the file on disk (left) and in the Project (right). This is how Subversion indicates that a file has modifications that have not been added to the repository.

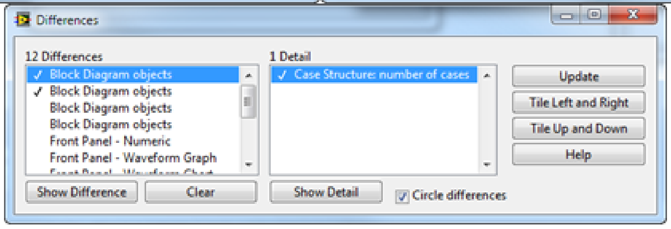


    d. We can compare the changes we've made with the latest version in source code control using Graphical Differencing. If the Viewpoint TortoiseSVN Plugin is installed, simply right-click on the file and select **Diff.** This can also be done directly from the TortoiseSVN client in Windows Explorer



    e. This will launch LVCompare.exe, showing a side-by-side comparison of objects on the front panel and the block diagram.


Main.vi-revBASE.svn000.tmp.vi Block Diagram on Software Validation Demo.lvpr...
Main.vi Block Diagram on Software Validation Demo.lvproj/My Computer
Main Application Loop
"Analyze Input Data"
[Covers: SoftwareReq3]
Initialize
Inflate
Original Signal
Rate Params
Timming
True
Process Heartbeat Waveform.vi
Enable Automated
error in (no error)
Software Validation Demo.lvproj/My Computer

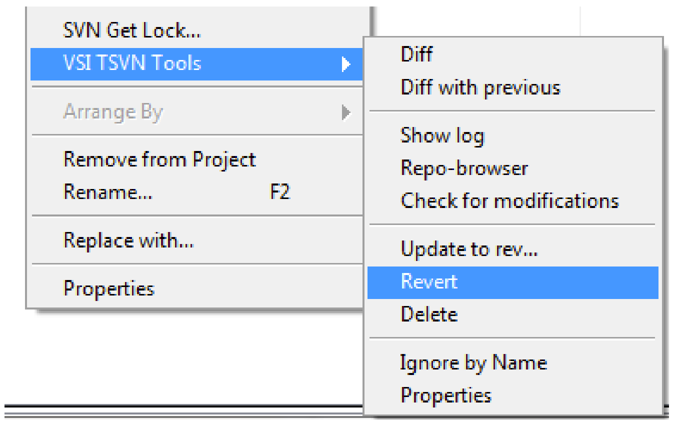


f. Double clicking on the items in the list will place a check mark next to them, indicating that you have examined and reviewed every change.

g. Click the 'X' in the 'Differences' window to close the dialog

**Note:** As of LabVIEW 2013, it is now possible to export the results of a diff operation to a file for external review. Use this dialog, select **Create Report** and select the file type preferred

2. Revert to the unmodified version we started with
   a. Make sure that the front panel of main.vi has been closed
   b. Right-click on the VI in the Project Explorer and select 'VSI TSVN Tools > Revet' to undo all of the changes that have been made



   c. Reopen main.vi and observe that it has reverted to the original copy of code

## Tracking Requirements Coverage

Most engineering projects start with high-level specifications, followed by the definition of more detailed specifications as the project progresses. Specifications contain technical and procedural requirements that guide the product through each engineering phase. In addition, working documents, such as hardware schematics, simulation models, software source code, and test specifications and procedures must adhere to and cover the requirements defined by specifications.

Requirements gathering is important in order to ensure that you and your customer have come to the same agreement about what the application will do. The granularity of the documents directly depends upon the needs of your application and the criticality of it. For mission-critical systems, it's typical to go as far as to define the requirements for individual modules of code, code units, and even the tests for those units. Part of this process requires having reached an agreement of what is expected behavior and how the system should perform under any and all conditions.

Nebulous or vague specifications for a project can lead to a result that does not meet customer expectations. Consider an example where you are asked to build an automobile, but given no additional information. It's unlikely that the finished product would resemble what the customer had in mind. They may have expected a two-door car with a sunroof, but you built a convertible. Even in scenarios where they aren't required, insisting on extensive documentation of requirements, complemented by reviews of proof of concepts and prototypes, greatly increases a project's likelihood of success.

Prototyping and proof of concepts are a very important step towards developing requirements. It can be very hard to account for all contingencies and foresee all the ways in which the software will behave. Proof of concepts are also extremely valuable because they give the end-user or customer a feel for what the product will do, which helps developers and users come to a consensus. It is largely this principle upon which the Agile development method was derived, which emphasizes repeated and frequent iterations between development.

One of the biggest challenges of development, in any language, is tracing the implementation to the requirement or specification it was supposed to fulfill. From a project management standpoint this is important in order to gain insight into how far into the project you are. When requirements change, it is also valuable to have record of what other specifications or the implementation covering them may also be affected.

The software industry has a wide variety of tools at their disposal for managing specifications and requirements. Common tools include Telelogic DOORS and Requisite Pro. National Instruments provides a tool to automate integration with these products, called NI Requirements Gateway. NI Requirements Gateway facilitates the tracking of requirements coverage for these three types of documents.

# Exercise 4: Tracing Code to Requirements Documents

## Goal

Developers who have been given or defined requirements should be able to document when and where the requirements are covered in their application to show that they have done what they were supposed to do. Our goal is to track and understand the percentage of the requirements that have been met and where. We also need to be able to create traceability matrices and other forms of documentation.

## Scenario

We've been given requirements for a simple application – we need to document that we've implemented it.

## Description

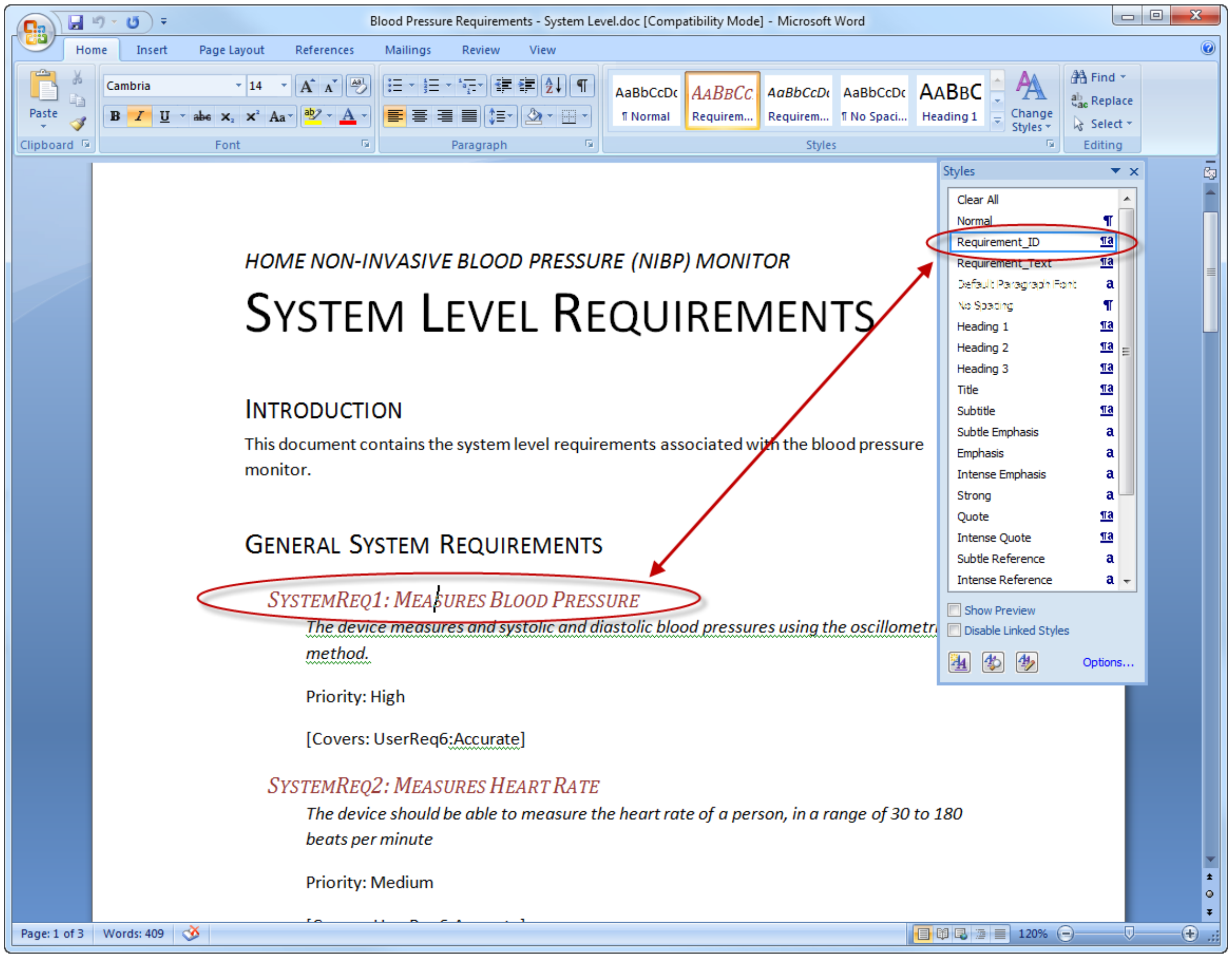
We are going to use NI Requirements Gateway to parse requirements documents written in Microsoft Word and generate reports. *Keep in mind that requirements could also be stored in DOORS, Requisite Pro, Excel, PDF, any many other standard formats.*

## Concepts Covered

- Adding a new item to a Requirements Gateway project
- Documenting code and requirements coverage
- Tracking requirements coverage percentage
- Generate traceability matrices and documentation

## INSTRUCTIONS

1. Document a new function in an application as having covered a requirement
   a. Open the requirements document, 'Blood Pressure Requirements – System Level' in Microsoft Word from the 'Requirements Documents' folder in the Project Explorer.
   b. Familiarize yourself with this simple requirements document. Note that these requirements are extremely high-level (and therefore difficult, if not impossible to test against or to 'cover' with an implementation). As a result, it will be necessary to use these high-level requirements to derive lower-level, more specific requirements.



*HOME NON-INVASIVE BLOOD PRESSURE (NIBP) MONITOR*

SYSTEM LEVEL REQUIREMENTS

INTRODUCTION

This document contains the system level requirements associated with the blood pressure monitor.

GENERAL SYSTEM REQUIREMENTS

*SYSTEMREQ1: MEASURES BLOOD PRESSURE*

*The device measures and systolic and diastolic blood pressures using the oscillometr method.*

Priority: High

[Covers: UserReq6:Accurate]

*SYSTEMREQ2: MEASURES HEART RATE*

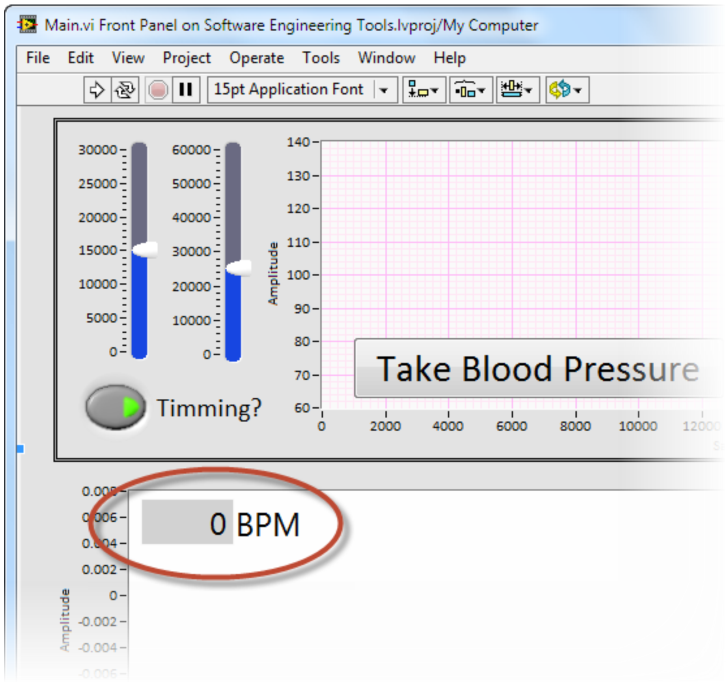
*The device should be able to measure the heart rate of a person, in a range of 30 to 180 beats per minute*

Priority: Medium

   c. Select one with your cursor and click on 'Styles' in the ribbon to observe that this text has been selected as a Requirements_ID. This will be used to automate the parsing of this document in later steps.
   d. Return to the folder containing the requirements document and open 'Detailed Software Design.docx.' This contains very specific requirements for the implementation and design of the software, which will actually be covered by the implementation in code. Note that it's divided into two main sections: State Implementations and GUI Components
   e. Scroll down to the last section, on GUI Component Requirements (page 12). In this scenario, we were given the requirements and asked to implement a UI component for

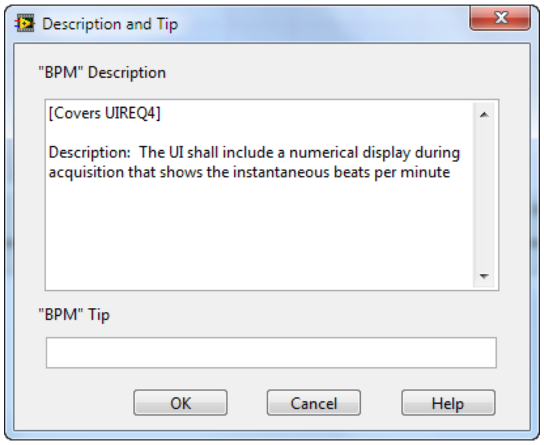
the BPM LED Indicator. The requirement as stated is, "The UI shall include a numerical display during acquisition that shows the instantaneous beats per minute." **Optional**: Copy the description of this requirement to the clipboard by highlighting it and pressing **CTRL+C**.

f. Open 'Software Validation Demo.lvproj' and open the Front Panel of 'Main.vi.' This version of the code has the required function successfully implemented for the BPM Indicator (circled below), we just need to document it.


Main.vi Front Panel on Software Engineering Tools.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Amplitude
Take Blood Pressure
Timming?
0 BPM

g. To document that this functionality has been implemented, right-click on the border of the indicator and select **Description and Tip**. This is where we will place the appropriate tag such that we can automatically parse and trace the relationship between this component and the actual implementation.

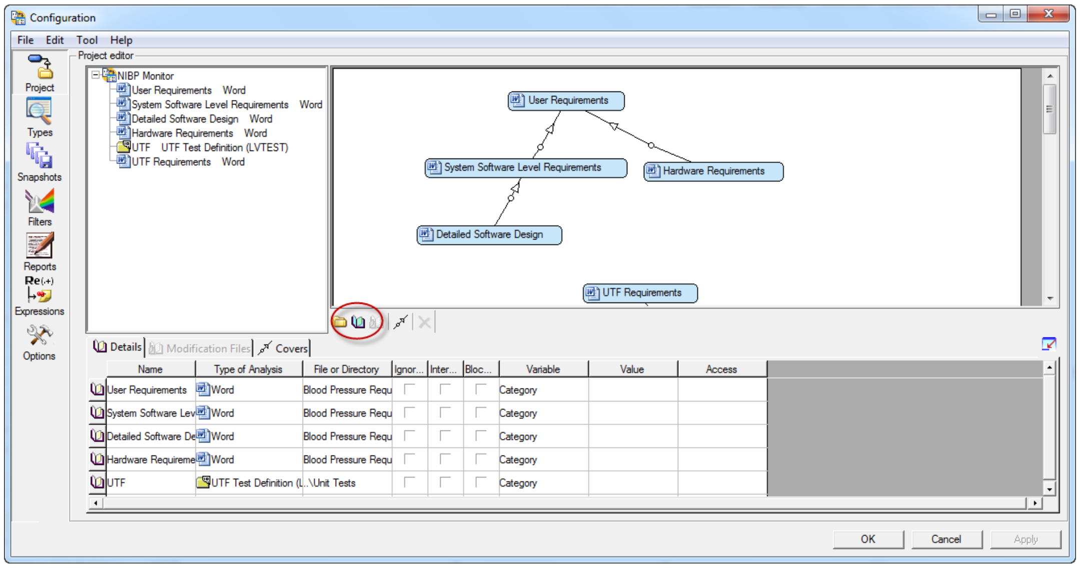
h. In the description field, type "[Covers: UIReq4]." Note that you should also include any other relevant information in this field (if you copied the text from step e, you can paste it in here)



[Covers UIREQ4]

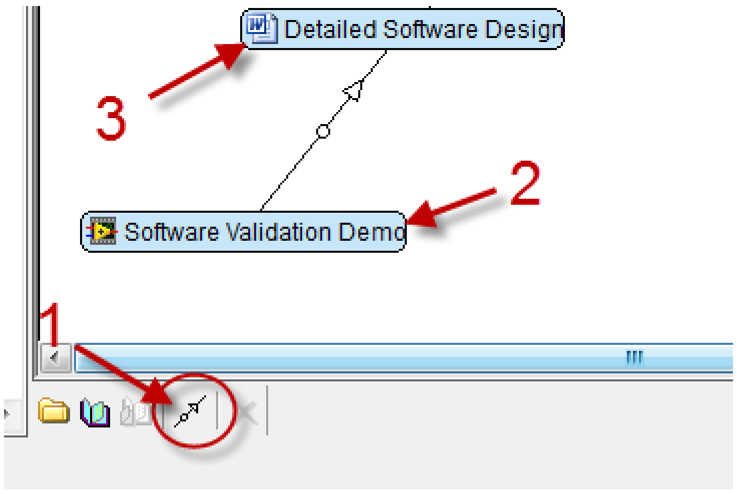
Description: The UI shall include a numerical display during acquisition that shows the instantaneous beats per minute

   i. Close the Description and Tip dialog by clicking **OK**
   j. Save the VI by pressing **CTRL + S**
2. Add the LabVIEW project to Requirements Gateway and parse for covered items
   a. Return to the LabVIEW Project Explorer and expand 'Requirements Documents'
   b. Double-click 'NIBP Monitor.rqtf,' which will open NI Requirements Gateway.
   c. The NI Requirements Gateway project has already been partially configured and shows the relationship between requirements documents, but does currently contain the LabVIEW code.
   d. Once NI Requirements Gateway is open, select edit **Edit Project** from the File menu
   e. Click on **Add Document** (circled below) and place a new container in the top-right window



| Name | Type of Analysis | File or Directory | Ignor... | Inter... | Bloc... | Variable | Value | Access |
|---|---|---|---|---|---|---|---|---|
| User Requirements | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| System Software Lev | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| Detailed Software De | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| Hardware Requireme | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| UTF | UTF Test Definition (L | ..\Unit Tests | ☐ | ☐ | ☐ | Category | | |

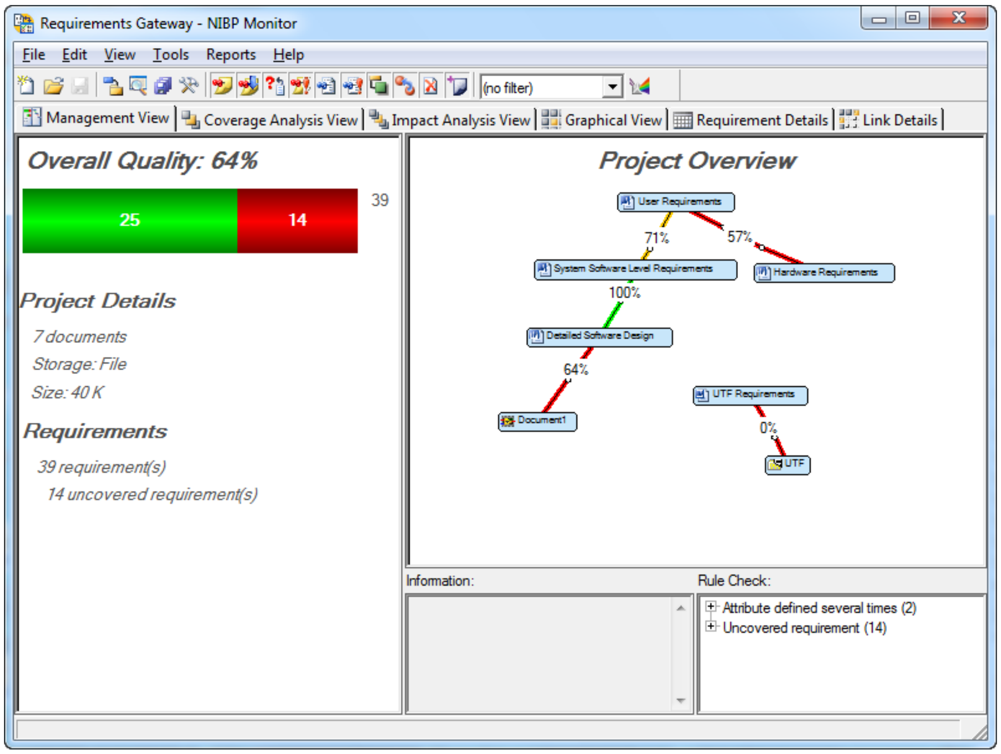
   f. Once placed, change the drop-down under 'type' to LabVIEW
   g. Click on **File or Directory** and click '…'
   h. In the dialog that appears, click **Add LabVIEW File…** and browse to the .lvproj file.
   i. Click **OK** to exit this dialog.
   j. Establish that the LabVIEW code covers the 'Detailed Software Design' requirements document. Click on **Add a cover** from the menu (shown below) and click on the LabVIEW document, then draw a line to the 'Detailed Software Design' document to indicate what this application is covering.



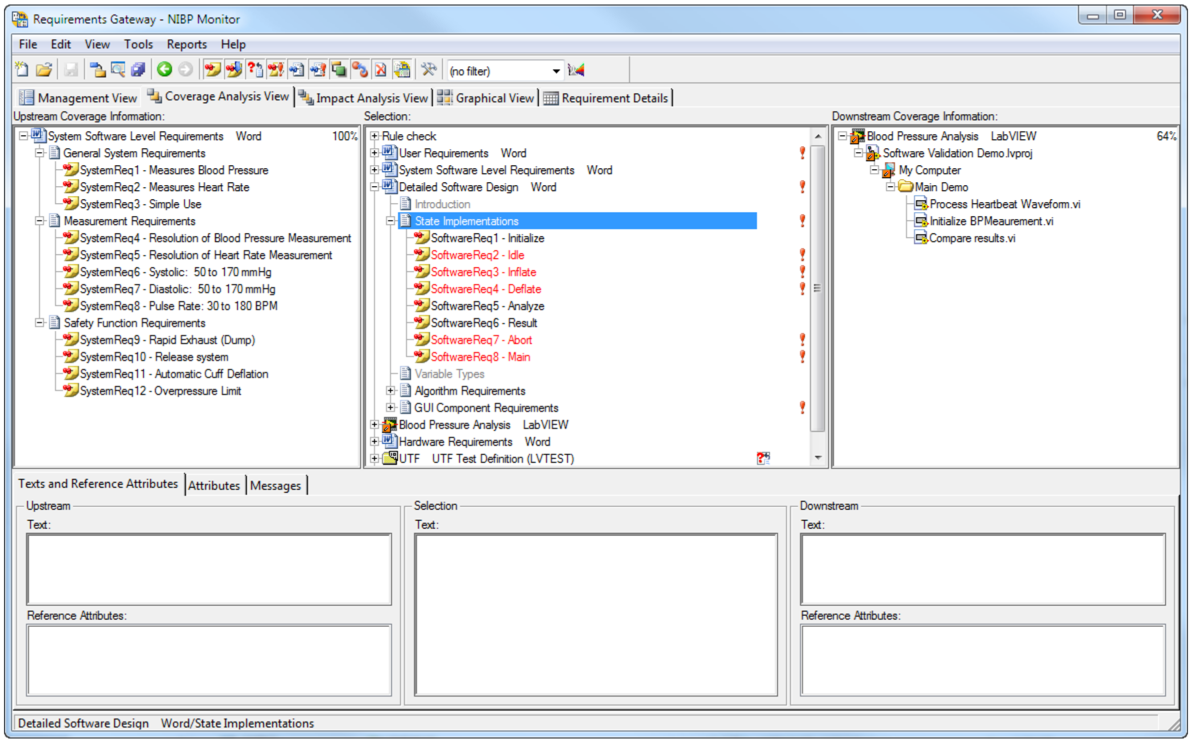
k. Click **OK** to exit the configuration dialog. If prompted to re-analyze the project, press **yes**.

l. Press **CTRL + S** to save the Requirements Gateway project.

m. In the Management View Tab, verify that Requirements Gateway is parsing these items as shown below:



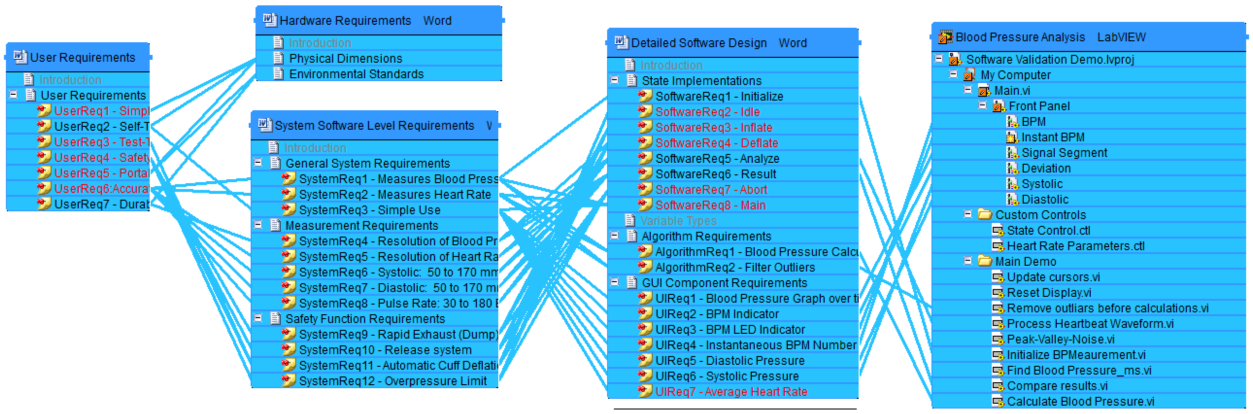
n. Notice that requirements coverage is less than 100%. Click on the **Coverage Analysis View** to see the list of uncovered requirements.



```
Requirements Gateway - NIBP Monitor
File  Edit  View  Tools  Reports  Help
(no filter)
Management View | Coverage Analysis View | Impact Analysis View | Graphical View | Requirement Details

Upstream Coverage Information:
⊟ System Software Level Requirements   Word   100%
   ⊟ General System Requirements
      SystemReq1 - Measures Blood Pressure
      SystemReq2 - Measures Heart Rate
      SystemReq3 - Simple Use
   ⊟ Measurement Requirements
      SystemReq4 - Resolution of Blood Pressure Measurement
      SystemReq5 - Resolution of Heart Rate Measurement
      SystemReq6 - Systolic:  50 to 170 mmHg
      SystemReq7 - Diastolic:  50 to 170 mmHg
      SystemReq8 - Pulse Rate: 30 to 180 BPM
   ⊟ Safety Function Requirements
      SystemReq9 - Rapid Exhaust (Dump)
      SystemReq10 - Release system
      SystemReq11 - Automatic Cuff Deflation
      SystemReq12 - Overpressure Limit

Selection:
⊞ Rule check
⊞ User Requirements   Word
⊞ System Software Level Requirements   Word
⊟ Detailed Software Design   Word
   Introduction
   ⊟ State Implementations
      SoftwareReq1 - Initialize
      SoftwareReq2 - Idle
      SoftwareReq3 - Inflate
      SoftwareReq4 - Deflate
      SoftwareReq5 - Analyze
      SoftwareReq6 - Result
      SoftwareReq7 - Abort
      SoftwareReq8 - Main
   Variable Types
   ⊞ Algorithm Requirements
   ⊞ GUI Component Requirements
⊞ Blood Pressure Analysis   LabVIEW
⊞ Hardware Requirements   Word
⊞ UTF   UTF Test Definition (LVTEST)

Downstream Coverage Information:
⊟ Blood Pressure Analysis   LabVIEW   64%
   ⊟ Software Validation Demo.lvproj
      ⊟ My Computer
         ⊟ Main Demo
            Process Heartbeat Waveform.vi
            Initialize BPMeaurement.vi
            Compare results.vi

Texts and Reference Attributes | Attributes | Messages
Upstream            Selection            Downstream
Text:               Text:                Text:
Reference Attributes:                    Reference Attributes:

Detailed Software Design   Word/State Implementations
```

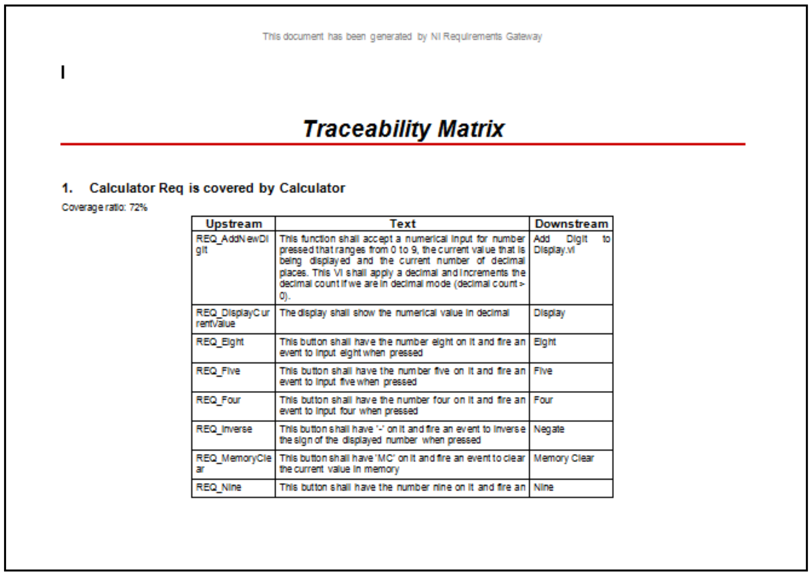
o. Click on the ‘Graphical View’ to see a graphical relationship between the requirements

3. Generate documentation showing requirements coverage
   a. In the graphical view, highlight what you want to include in the report. Hold **CTRL** while selecting both the requirements document, and the LabVIEW Project.



```
User Requirements
   Introduction
⊟ User Requirements
      UserReq1 - Simp
      UserReq2 - Self-T
      UserReq3 - Test-
      UserReq4 - Safet
      UserReq5 - Porta
      UserReq6:Accura
      UserReq7 - Durat

Hardware Requirements   Word
   Introduction
   Physical Dimensions
   Environmental Standards

System Software Level Requirements   W
   Introduction
⊟ General System Requirements
      SystemReq1 - Measures Blood Press
      SystemReq2 - Measures Heart Rate
      SystemReq3 - Simple Use
⊟ Measurement Requirements
      SystemReq4 - Resolution of Blood Pr
      SystemReq5 - Resolution of Heart Ra
      SystemReq6 - Systolic:  50 to 170 mm
      SystemReq7 - Diastolic:  50 to 170 m
      SystemReq8 - Pulse Rate: 30 to 180 E
⊟ Safety Function Requirements
      SystemReq9 - Rapid Exhaust (Dump)
      SystemReq10 - Release system
      SystemReq11 - Automatic Cuff Deflati
      SystemReq12 - Overpressure Limit

Detailed Software Design   Word
   Introduction
⊟ State Implementations
      SoftwareReq1 - Initialize
      SoftwareReq2 - Idle
      SoftwareReq3 - Inflate
      SoftwareReq4 - Deflate
      SoftwareReq5 - Analyze
      SoftwareReq6 - Result
      SoftwareReq7 - Abort
      SoftwareReq8 - Main
   Variable Types
⊟ Algorithm Requirements
      AlgorithmReq1 - Blood Pressure Calc
      AlgorithmReq2 - Filter Outliers
⊟ GUI Component Requirements
      UIReq1 - Blood Pressure Graph over ti
      UIReq2 - BPM Indicator
      UIReq3 - BPM LED Indicator
      UIReq4 - Instantaneous BPM Number
      UIReq5 - Diastolic Pressure
      UIReq6 - Systolic Pressure
      UIReq7 - Average Heart Rate

Blood Pressure Analysis   LabVIEW
⊟ Software Validation Demo.lvproj
   ⊟ My Computer
      ⊟ Main.vi
         ⊟ Front Panel
               BPM
               Instant BPM
               Signal Segment
               Deviation
               Systolic
               Diastolic
      ⊟ Custom Controls
            State Control.ctl
            Heart Rate Parameters.ctl
      ⊟ Main Demo
            Update cursors.vi
            Reset Display.vi
            Remove outliars before calculations.vi
            Process Heartbeat Waveform.vi
            Peak-Valley-Noise.vi
            Initialize BPMeaurement.vi
            Find Blood Pressure_ms.vi
            Compare results.vi
            Calculate Blood Pressure.vi
```

   b. Click on **Reports > Library Reports > Traceability Matrix**
   c. Ensure that you’ve selected all the items to include and click **Continue**
   d. In the save dialogue that appears, note the different formats that are available. Select PDF and select the desktop and type ‘Blood Pressure Traceability Matrix’
   e. The traceability matrix will appear



This document has been generated by NI Requirements Gateway

I

## Traceability Matrix

### 1. Calculator Req is covered by Calculator

Coverage ratio: 72%

| Upstream | Text | Downstream |
|---|---|---|
| REQ_AddNewDigit | This function shall accept a numerical input for number pressed that ranges from 0 to 9, the current value that is being displayed and the current number of decimal places. This VI shall apply a decimal and increments the decimal count if we are in decimal mode (decimal count > 0). | Add Digit to Display.vi |
| REQ_DisplayCurrentValue | The display shall show the numerical value in decimal | Display |
| REQ_Eight | This button shall have the number eight on it and fire an event to input eight when pressed | Eight |
| REQ_Five | This button shall have the number five on it and fire an event to input five when pressed | Five |
| REQ_Four | This button shall have the number four on it and fire an event to input four when pressed | Four |
| REQ_Inverse | This button shall have '-' on it and fire an event to inverse the sign of the displayed number when pressed | Negate |
| REQ_MemoryClear | This button shall have 'MC' on it and fire an event to clear the current value in memory | Memory Clear |
| REQ_Nine | This button shall have the number nine on it and fire an | Nine |

## PERFORMING CODE REVIEWS

Regular and thorough code reviews are an important and common practice for software engineers seeking to mitigate the risk of unforeseen problems, identify the cause of bugs that are difficult to find, align the styles of multiple developers, and demonstrate that the code works. These reviews are an opportunity for a team of qualified individuals to scrutinize the logic of the developer and analyze the performance of the software.

Peer reviews are sometimes referred to as a code 'walk-through.' The reviewer is typically guided through the main path of execution through the program by the developer, during which they should be examining the programming style, checking for adequate documentation, and reviewing questions that can be common stumbling blocks, such as:

- How easily can new features be added in the future?
- How are errors reported and handled?
- Is the code modular enough?
- Does the code starve the processor or use a prohibitive amount of memory?
- Is an adequate testing plan in place?

One of the most common reasons for not performing a code review is the amount of time needed to prepare for and then perform the review. In order to simplify the process, you need to take advantage of tools that can help automate the code inspection and help identify improvements. One example is the LabVIEW VI Analyzer tool, which analyzes any LabVIEW code and then steps the user through the test failures. You can also generate reports that allow you track code improvements over time, and can be checked into source code control software along with your VIs.

# Exercise 5: Analyzing Code Quality

## Goal

We want to analyze our code on a regular basis to identify any potential problems or coding errors that could cause inappropriate or incorrect behavior.

## Scenario

We're going to be configuring a series of tests, examining the results, and generating a report to document the results.

## Description

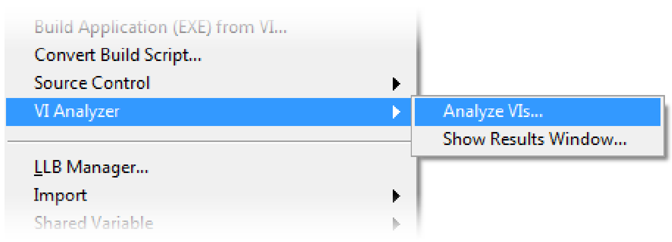
The NI LabVIEW VI Analyzer Toolkit will be used to run 90+ tests on our application hierarchy and generate an HTML report.

## Concepts Covered

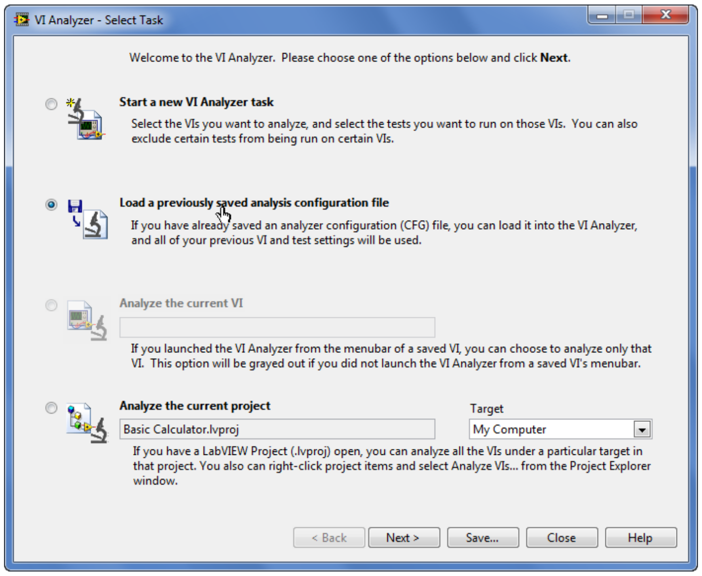
- Loading pre-configured test configuration
- Report generation

## INSTRUCTIONS

1. Launch, configure and run the analyzer tests
   a. From within LabVIEW, select **Tools > VI Analyzer > Analyze VIs…**



   b. Select the task labeled **Load a previously saved analysis configuration file** and click **Next**


VI Analyzer - Select Task

Welcome to the VI Analyzer. Please choose one of the options below and click **Next**.

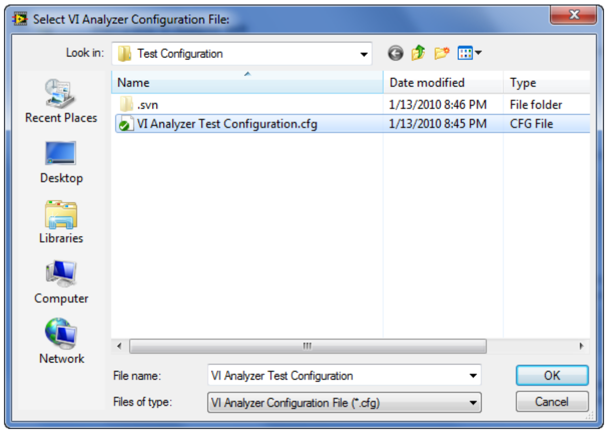
**Start a new VI Analyzer task**
Select the VIs you want to analyze, and select the tests you want to run on those VIs. You can also exclude certain tests from being run on certain VIs.

**Load a previously saved analysis configuration file**
If you have already saved an analyzer configuration (CFG) file, you can load it into the VI Analyzer, and all of your previous VI and test settings will be used.

Analyze the current VI
If you launched the VI Analyzer from the menubar of a saved VI, you can choose to analyze only that VI. This option will be grayed out if you did not launch the VI Analyzer from a saved VI's menubar.

**Analyze the current project**
Basic Calculator.lvproj
Target: My Computer
If you have a LabVIEW Project (.lvproj) open, you can analyze all the VIs under a particular target in that project. You also can right-click project items and select Analyze VIs... from the Project Explorer window.

   c. VI Analyzer allows us to customize test settings and save the configuration for future use. Navigate to ‘Software Validation Demo > Code Reviews,’ and load the **VI Analyzer Test Configuration.cfg** file.

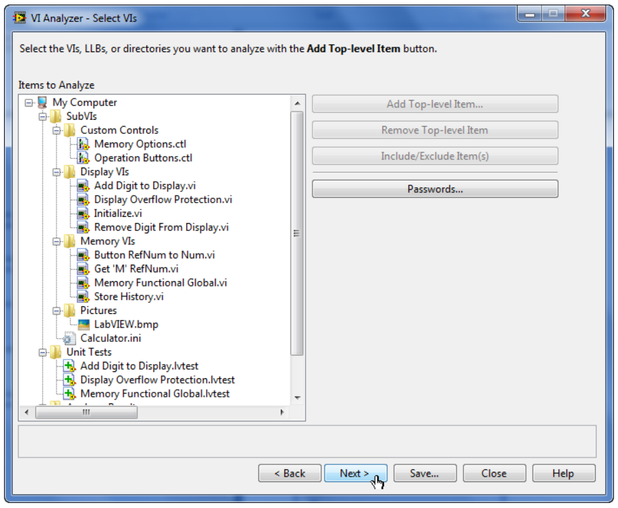

Select VI Analyzer Configuration File:

Look in: Test Configuration

| Name | Date modified | Type |
|---|---|---|
| .svn | 1/13/2010 8:46 PM | File folder |
| VI Analyzer Test Configuration.cfg | 1/13/2010 8:45 PM | CFG File |

File name: VI Analyzer Test Configuration
Files of type: VI Analyzer Configuration File (*.cfg)

d. VI Analyzer will display a list of files that will be analyzed. We can use this window to add or remove objects. Since all the files we want to analyze have been selected, click **Next**.


VI Analyzer - Select VIs

Select the VIs, LLBs, or directories you want to analyze with the **Add Top-level Item** button.

Items to Analyze

```
My Computer
  SubVIs
    Custom Controls
      Memory Options.ctl
      Operation Buttons.ctl
    Display VIs
      Add Digit to Display.vi
      Display Overflow Protection.vi
      Initialize.vi
      Remove Digit From Display.vi
    Memory VIs
      Button RefNum to Num.vi
      Get 'M' RefNum.vi
      Memory Functional Global.vi
      Store History.vi
    Pictures
      LabVIEW.bmp
    Calculator.ini
  Unit Tests
    Add Digit to Display.lvtest
    Display Overflow Protection.lvtest
    Memory Functional Global.lvtest
```

e. A list of all available tests will be displayed. Select a test to view configuration information and set the priority. Recommendations include:

1. Documentation > User > **Spell Check** – this can help mitigate the risk of misspelled words amongst the documentation and most importantly, the user interface
2. Complexity Metrics > **Cycolmatic Complexity** – this industry-standard code metric helps evaluate the amount of paths through code, which is useful when developing test plans.
3. Block Diagram > Performance > **Arrays and Strings in Loops** – this is one of several tests that can point out programming practices that could detract from execution speed.

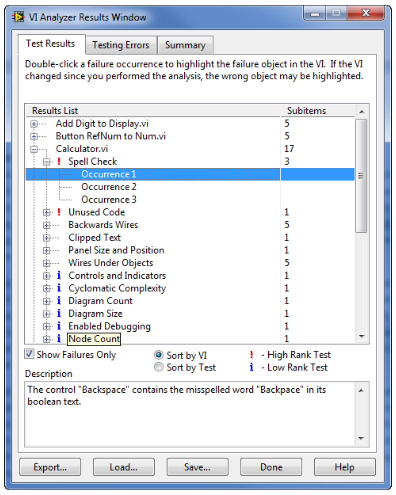
f. Click **Next**

g. We can now save the test configuration, or we can perform the analysis on our VIs. Click **Analyze** to begin testing the entire hierarchy of VIs.

Note: The tests should take roughly thirty seconds to complete on a fast computer if you run them on the entire application hierarchy.

2. Review the VI Analyzer Results and Correct Errors

a. The dialog that appears after running the tests shows the list of VIs that was analyzed. The number shown in the right column indicates the number of items that require attention and review. Begin by expanding the items under 'main.vi.'



VI Analyzer Results Window

Test Results | Testing Errors | Summary

Double-click a failure occurrence to highlight the failure object in the VI. If the VI changed since you performed the analysis, the wrong object may be highlighted.

| Results List | Subitems |
|---|---|
| Add Digit to Display.vi | 5 |
| Button RefNum to Num.vi | 5 |
| Calculator.vi | 17 |
| ! Spell Check | 3 |
| Occurrence 1 | |
| Occurrence 2 | |
| Occurrence 3 | |
| ! Unused Code | 1 |
| Backwards Wires | 5 |
| Clipped Text | 1 |
| Panel Size and Position | 1 |
| Wires Under Objects | 5 |
| i Controls and Indicators | 1 |
| i Cyclomatic Complexity | 1 |
| i Diagram Count | 1 |
| i Diagram Size | 1 |
| i Enabled Debugging | 1 |
| i Node Count | 1 |

☑ Show Failures Only

● Sort by VI
○ Sort by Test

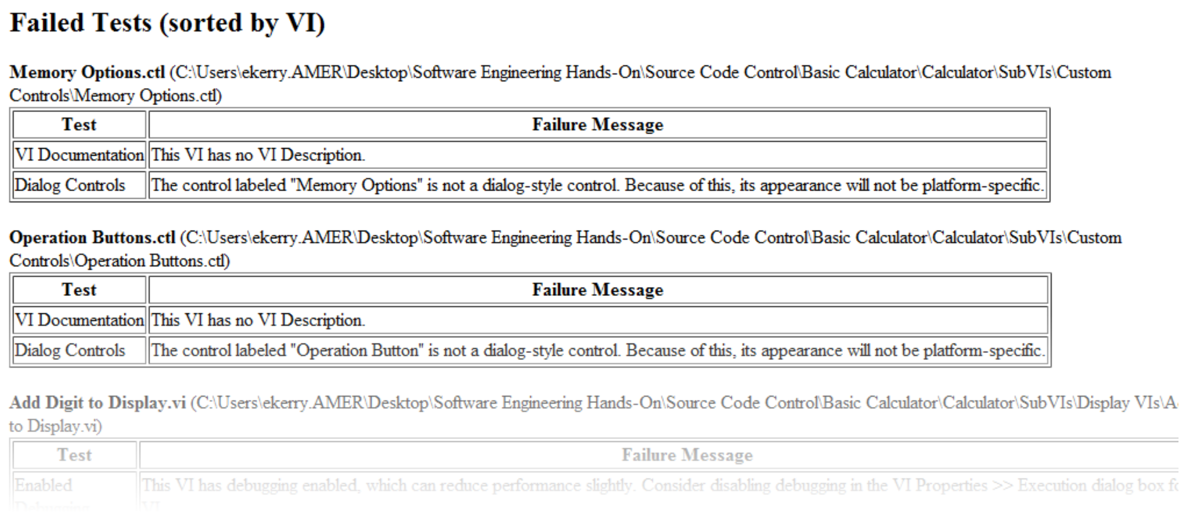
! - High Rank Test
i - Low Rank Test

Description

The control "Backspace" contains the misspelled word "Backpace" in its boolean text.

b. The high importance test failures will be indicated using a red exclamation mark. As an example, expand the 'Spell Check' test and select 'Occurrence 1.' The description should explain that, 'The control "Timing" contains the misspelled word "Timing" in its Boolean text.'
c. Double-click on Occurrence 1. LabVIEW should highlight the button on the front panel with the misspelled word.
d. Explore the remaining results and consult the description for details on how to correct the error.

3. Generate an HTML report
    a. Click 'Export' in the VI Analyzer Results Window.
    b. Change the location to the 'Code Review' folder in the project and type in the name of the file you wish to save it as.
    c. From the drop-down, select **HTML**.
    d. Click **Export**
    e. Click **Done** on the VI Analyzer Results Window. Click **No** to dismiss the save dialog.
    f. When prompted to return to VI Analyzer, click **No**
    g. From within the Project Explorer, expand the 'Code Review' folder
    h. Double-click the new html document to see the results in a browser. Note that this dialog includes links to tests for navigation.



## Failed Tests (sorted by VI)

**Memory Options.ctl** (C:\Users\ekerry.AMER\Desktop\Software Engineering Hands-On\Source Code Control\Basic Calculator\Calculator\SubVIs\Custom Controls\Memory Options.ctl)

| Test | Failure Message |
|---|---|
| VI Documentation | This VI has no VI Description. |
| Dialog Controls | The control labeled "Memory Options" is not a dialog-style control. Because of this, its appearance will not be platform-specific. |

**Operation Buttons.ctl** (C:\Users\ekerry.AMER\Desktop\Software Engineering Hands-On\Source Code Control\Basic Calculator\Calculator\SubVIs\Custom Controls\Operation Buttons.ctl)

| Test | Failure Message |
|---|---|
| VI Documentation | This VI has no VI Description. |
| Dialog Controls | The control labeled "Operation Button" is not a dialog-style control. Because of this, its appearance will not be platform-specific. |

**Add Digit to Display.vi** (C:\Users\ekerry.AMER\Desktop\Software Engineering Hands-On\Source Code Control\Basic Calculator\Calculator\SubVIs\Display VIs\A to Display.vi)

| Test | Failure Message |
|---|---|
| Enabled Debugging | This VI has debugging enabled, which can reduce performance slightly. Consider disabling debugging in the VI Properties >> Execution dialog box f VI. |

## ADVANCED DEBUGGING AND DYNAMIC CODE ANALYSIS

Identifying the source and fixing the cause of unexpected or undesirable behavior in software can be a tedious, time-consuming and expensive task for developers. Even code that is syntactically correct and functionally complete is often still contaminated with problems such as memory leaks or daemon tasks that can impact performance or lead to incorrect behavior. These oversights can be difficult to reproduce and even more difficult to locate, especially in large, complex applications.

With the NI LabVIEW Desktop Execution Trace Toolkit, we can trace the execution of LabVIEW VIs on a Windows target during run-time to detect and locate problems in code that could impact performance or cause unexpected behavior. This will be helpful when struggling to locate the source of difficult to find, or difficult to reproduce issues. The Desktop Execution Trace Toolkit provides a chronological view of system events, queue operations, reference leaks, memory allocation, un-handled errors, and the execution of subVIs. Users can also programmatically generate user-defined events from the block diagram of a LabVIEW application.

Dynamic code analysis refers to the ability to understand what software is doing ‘under-the-hood’ during execution. In other words, it provides details about events and the context in which they occur in order to give developers a bigger picture and more information that can help solve problems.

Dynamic code analysis has a number of different use-cases throughout the software development life-cycle, including:

- Detecting memory and reference leaks
- Isolating the source of a specific event or undesired behavior
- Screening applications for areas where performance can be improved
- Identifying the last call before an error
- Ensuring the execution of an application is the same on different targets

Problems such as memory leaks can have costly consequences for systems that are required to sustain operation for extended periods of time or for software that has been released to a customer. If software that needs debugging has been deployed and the LabVIEW development environment is not installed on the current machine, it may be beneficial to perform dynamic analysis of the code with the Desktop Execution Trace Toolkit over the network. For deployed systems, even if the development environment is available, it may be impractical or difficult to locally troubleshoot or profile the execution of a running system.

# Exercise 6: Debugging Unexpected Behavior

## Goal

We want to profile the execution of a LabVIEW application we've developed to find the source of un-desirable behavior.

## Scenario

Consider that you have software in use that appears to work fine, but it eventually begins to get slower and less responsive, or eventually quits unexpectedly. You suspect a memory leak, but the application is very large and it could take an extremely long time to track down the source of this problem.

## Description

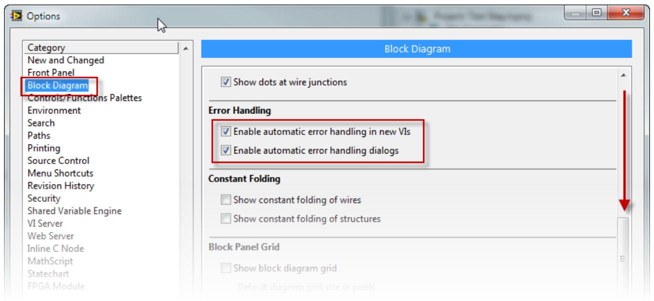
We're going to use the Desktop Execution Trace Toolkit to monitor the execution of our suspect application and see if we can find the source of these problematic behaviors.

## Concepts Covered

- How to setup and configure a trace
- How to filter the information
- User-defined trace data
- Finding the source of an event
- Identifying memory leaks
- Discovering un-handled errors

## Setup

- Make sure that LabVIEW and the Desktop Execution Trace Toolkit are installed.
- Make sure that a firewall is not preventing communication between the tool and LabVIEW
- In LabVIEW, go to Tools >> Options and select the 'Block Diagram' category. De-select **Enable Automatic Error Handling in new VIs** and de-select '**Enable Automatic Error handling dialogs**.



[✓] Show dots at wire junctions

Error Handling

[✓] Enable automatic error handling in new VIs

[✓] Enable automatic error handling dialogs

Constant Folding

[ ] Show constant folding of wires

[ ] Show constant folding of structures

Block Panel Grid

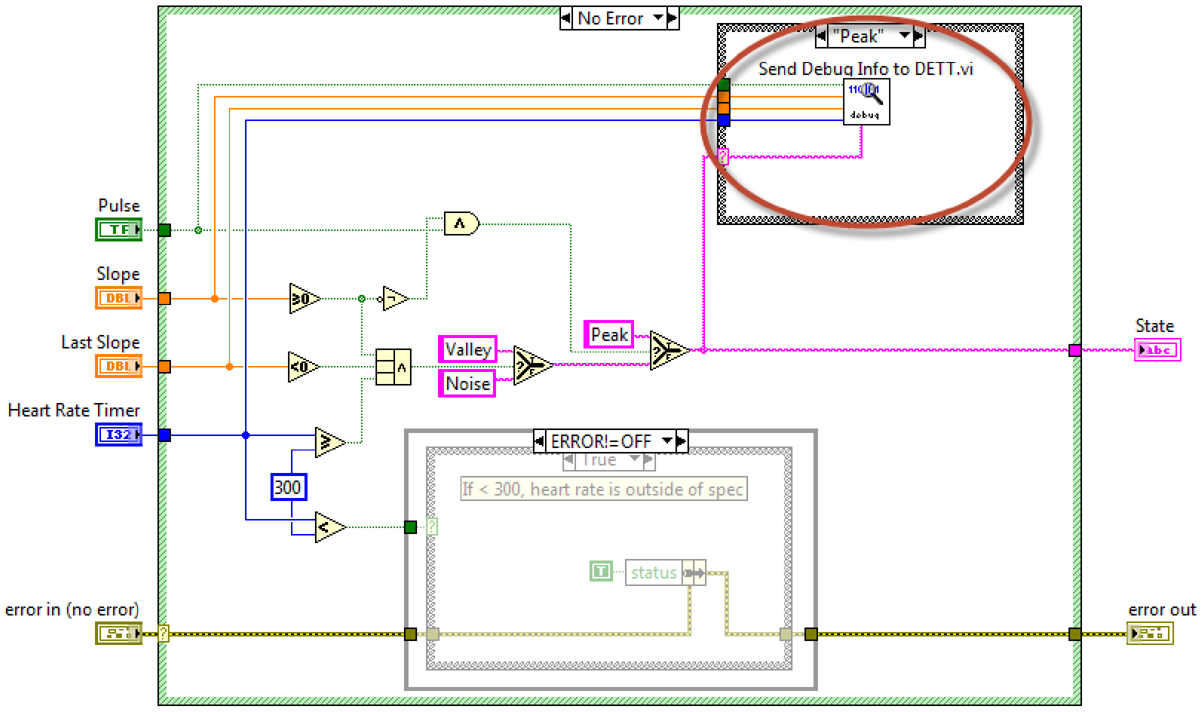
[ ] Show block diagram grid

Default diagram grid size in pixels

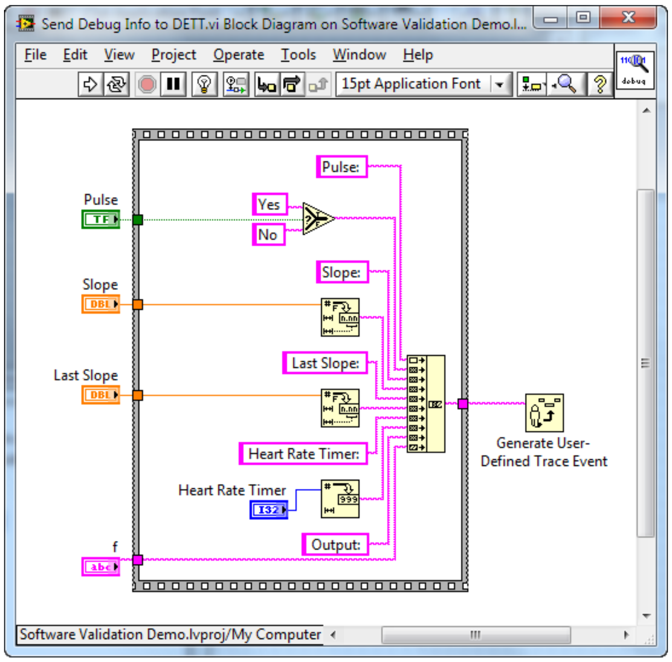
- To identify the appropriate VI Server port to use, click 'Help > About LabVIEW…' from the file menu. On the screen that appears, it will be identified as the port on which the TCP server is active, or 'TCP Server active on port ####'

## INSTRUCTIONS

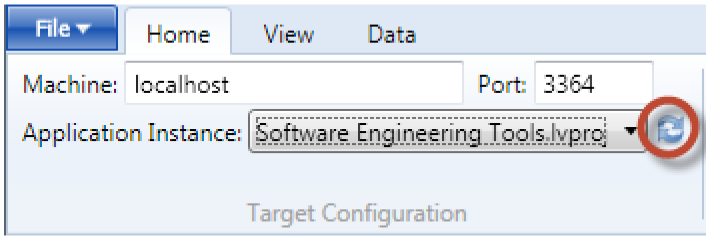
1. Demonstrate that application does not appear to have any obvious defects and show how custom trace data has been programmed into the application
   a. In the Project Explorer, open the VI entitled **Main.vi**. As has been demonstrated in prior exercises, the application works correctly and does not have any obvious bugs; however, errors have been intentionally coded into this application for the sake of demonstration, including a memory leak that will degrade performance over time.
   b. In the Project Explorer, expand the folder 'subVIs' and open the VI 'Peal-Valley-Noise.vi.' This VI is responsible to isolating peaks and valleys from the filtered data – if, as an example, this VI was incorrectly reporting a noise as a valley, we might notice that the mean HR is slightly off as a result. In the course of debugging this problem, it would be helpful to know all of the parameters when this incorrect computation is being made. Setting a breakpoint would be impractical because this VI is run for every unique point in the waveform, which means a single acquisition may execute this VI upwards of 10,000 times in just a few seconds. If the VI only returns incorrect results intermittently, it's much more useful to be able to record execution and all of the parameters for review afterwards. To accomplish this, all of the parameters used in this calculation are based to 'Send Debug Info to DETT' when a Valley is detected.



   c. Open the block diagram of 'Send Debug Info to DETT.vi' to see how the information is packaged. This VI uses the 'Generate User-Defined Trace Event' primitive to stream data to the Desktop Execution Trace, which is extremely helpful when trying to debug code that quickly iterates through a large amount of data. .

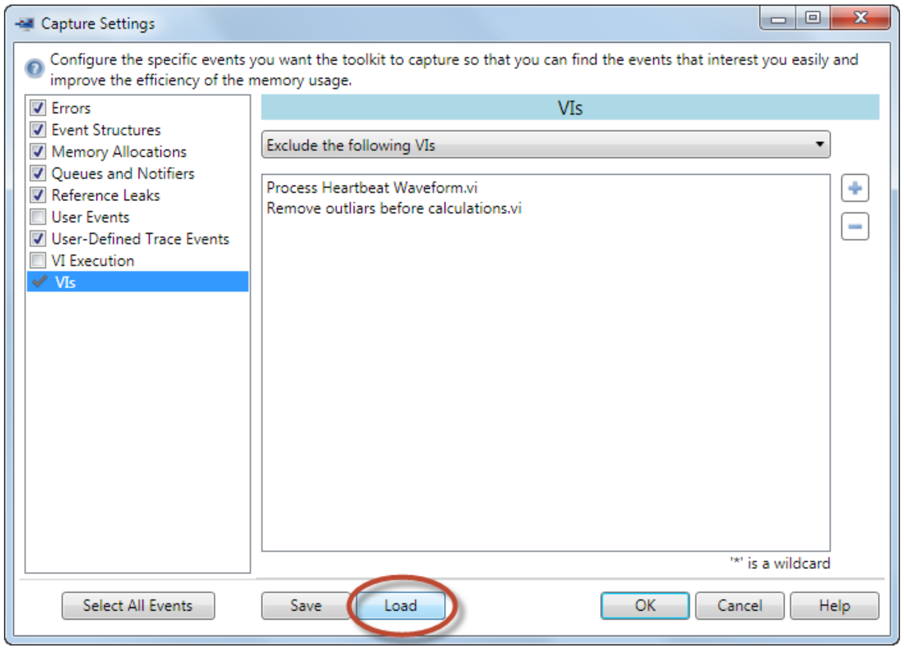

Send Debug Info to DETT.vi Block Diagram on Software Validation Demo.l...
File Edit View Project Operate Tools Window Help
15pt Application Font
Pulse
Slope
Last Slope
f
Pulse:
Yes
No
Slope:
Last Slope:
Heart Rate Timer:
Heart Rate Timer
Output:
Generate User-Defined Trace Event
Software Validation Demo.lvproj/My Computer

2. Setup the Trace
   a. Launch the Desktop Execution Trace Toolkit from the start menu by navigating to **National Instruments > LabVIEW Desktop Execution Trace Toolkit**
   b. On the **Home** tab in the ribbon, ensure that the correct port matches the VI Server port LabVIEW is using (see setup instructions) and click **refresh** (circled below) to load the application instances that are available.



Machine: localhost
Port: 3364
Application Instance: Software Engineering Tools.lvproj

   c. Select **Application Instance** you want to trace from the drop down – this should correspond to the name of the Project Explorer you want to trace.
   d. Click on **Capture Settings** to define what will be recorded
   e. This dialog can be used to customize the type of information you want to see. This tool can quickly return large amounts of information, so it's helpful to isolate your trace to key VIs and/or key information. Click **Load** on the bottom to load a previous trace configuration. Navigate to the location of the Project on disk and open the 'Code Reviews' folder. Open the 'Trace Configuration.ini' file.


Capture Settings

Configure the specific events you want the toolkit to capture so that you can find the events that interest you easily and improve the efficiency of the memory usage.

- [✓] Errors
- [✓] Event Structures
- [✓] Memory Allocations
- [✓] Queues and Notifiers
- [✓] Reference Leaks
- [ ] User Events
- [✓] User-Defined Trace Events
- [ ] VI Execution
- [✓] VIs

VIs

Exclude the following VIs

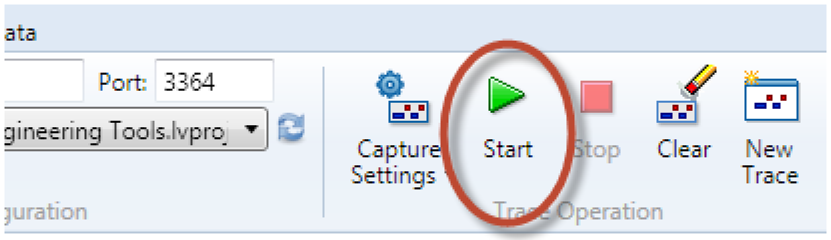
Process Heartbeat Waveform.vi
Remove outliars before calculations.vi

'*' is a wildcard

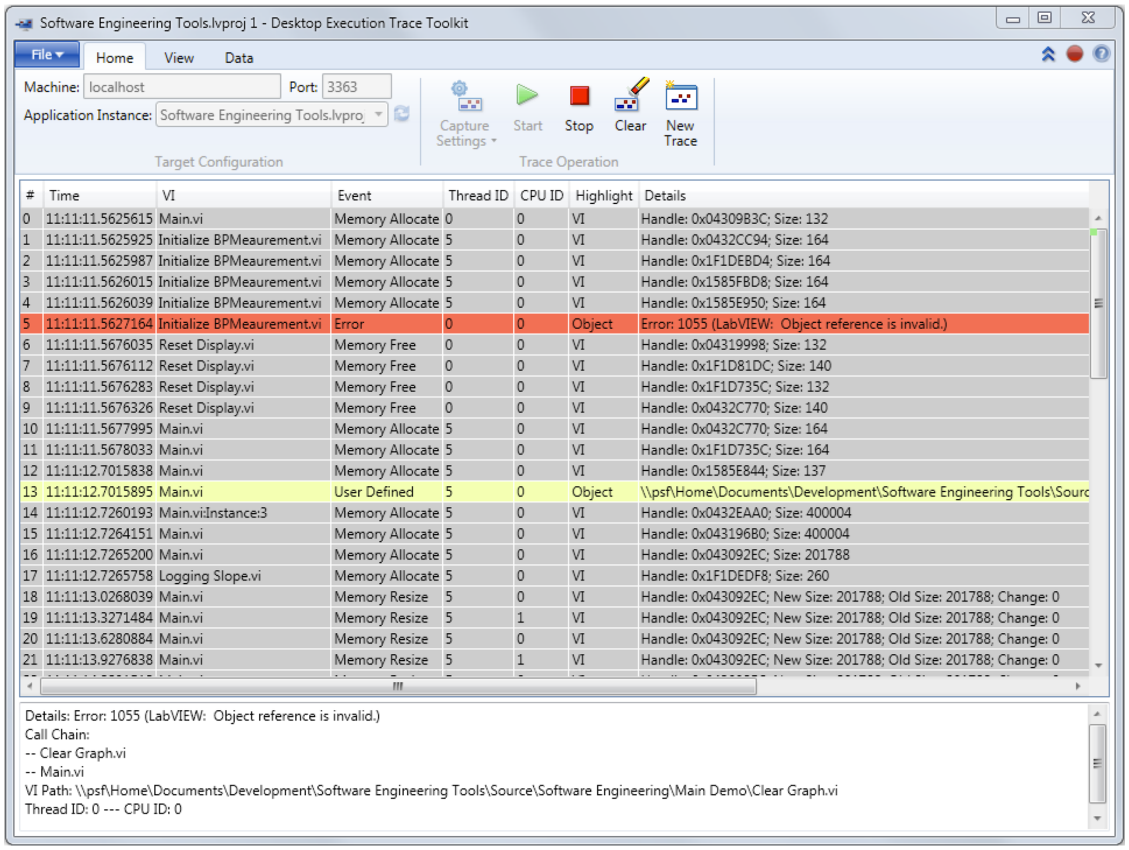
f. Click **OK**

**Note:** Feel free to repeat the exercise and experiment with different configurations. There are numerous examples in the Example Finder for the Desktop Execution Trace Toolkit that can be used to illustrate the tool's capabilities.

3. Begin tracing execution
   a. Select **Start** from the toolbar



   b. Switch back to the front panel of **Main.vi**
   c. Position the front panel and the Desktop Execution Trace Toolkit on the screen so that both can be seen simultaneously
   d. **Run** the VI and when prompted, select the same sample data as last time
   e. Note that you should see data appear in the trace as shown below after selecting the dataset.

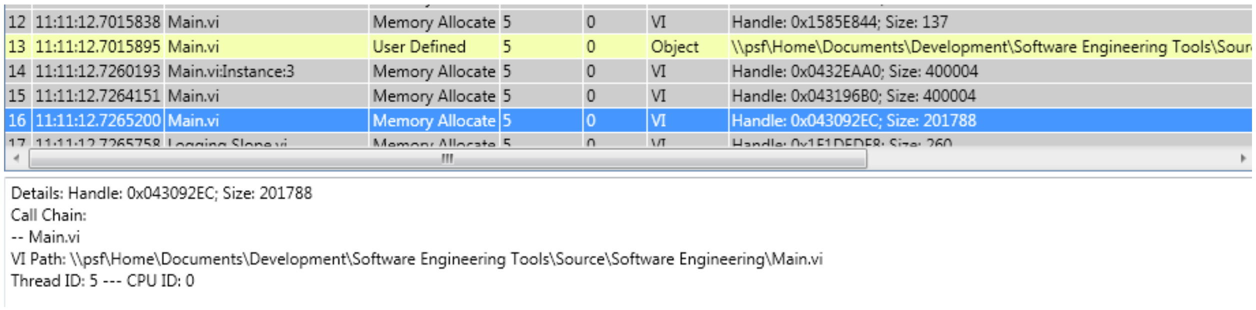

Software Engineering Tools.lvproj 1 - Desktop Execution Trace Toolkit
File
Home
View
Data
Machine: localhost
Port: 3363
Application Instance: Software Engineering Tools.lvproj
Target Configuration
Capture Settings
Start
Stop
Clear
New Trace
Trace Operation

| # | Time | VI | Event | Thread ID | CPU ID | Highlight | Details |
|---|---|---|---|---|---|---|---|
| 0 | 11:11:11.5625615 | Main.vi | Memory Allocate | 0 | 0 | VI | Handle: 0x04309B3C; Size: 132 |
| 1 | 11:11:11.5625925 | Initialize BPMeaurement.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x0432CC94; Size: 164 |
| 2 | 11:11:11.5625987 | Initialize BPMeaurement.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1F1DEBD4; Size: 164 |
| 3 | 11:11:11.5626015 | Initialize BPMeaurement.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1585FBD8; Size: 164 |
| 4 | 11:11:11.5626039 | Initialize BPMeaurement.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1585E950; Size: 164 |
| 5 | 11:11:11.5627164 | Initialize BPMeaurement.vi | Error | 0 | 0 | Object | Error: 1055 (LabVIEW: Object reference is invalid.) |
| 6 | 11:11:11.5676035 | Reset Display.vi | Memory Free | 0 | 0 | VI | Handle: 0x04319998; Size: 132 |
| 7 | 11:11:11.5676112 | Reset Display.vi | Memory Free | 0 | 0 | VI | Handle: 0x1F1D81DC; Size: 140 |
| 8 | 11:11:11.5676283 | Reset Display.vi | Memory Free | 0 | 0 | VI | Handle: 0x1F1D735C; Size: 132 |
| 9 | 11:11:11.5676326 | Reset Display.vi | Memory Free | 0 | 0 | VI | Handle: 0x0432C770; Size: 140 |
| 10 | 11:11:11.5677995 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x0432C770; Size: 164 |
| 11 | 11:11:11.5678033 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1F1D735C; Size: 164 |
| 12 | 11:11:12.7015838 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1585E844; Size: 137 |
| 13 | 11:11:12.7015895 | Main.vi | User Defined | 5 | 0 | Object | \\psf\Home\Documents\Development\Software Engineering Tools\Sourc |
| 14 | 11:11:12.7260193 | Main.vi:Instance:3 | Memory Allocate | 5 | 0 | VI | Handle: 0x0432EAA0; Size: 400004 |
| 15 | 11:11:12.7264151 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x043196B0; Size: 400004 |
| 16 | 11:11:12.7265200 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x043092EC; Size: 201788 |
| 17 | 11:11:12.7265758 | Logging Slope.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1F1DEDF8; Size: 260 |
| 18 | 11:11:13.0268039 | Main.vi | Memory Resize | 5 | 0 | VI | Handle: 0x043092EC; New Size: 201788; Old Size: 201788; Change: 0 |
| 19 | 11:11:13.3271484 | Main.vi | Memory Resize | 5 | 1 | VI | Handle: 0x043092EC; New Size: 201788; Old Size: 201788; Change: 0 |
| 20 | 11:11:13.6280884 | Main.vi | Memory Resize | 5 | 0 | VI | Handle: 0x043092EC; New Size: 201788; Old Size: 201788; Change: 0 |
| 21 | 11:11:13.9276838 | Main.vi | Memory Resize | 5 | 1 | VI | Handle: 0x043092EC; New Size: 201788; Old Size: 201788; Change: 0 |

```
Details: Error: 1055 (LabVIEW:  Object reference is invalid.)
Call Chain:
-- Clear Graph.vi
-- Main.vi
VI Path: \\psf\Home\Documents\Development\Software Engineering Tools\Source\Software Engineering\Main Demo\Clear Graph.vi
Thread ID: 0 --- CPU ID: 0
```

   f. The screen will continue to populate with 'Memory Resize' as we have code in our application that has a leak in it.
   g. Return to the application and click **Take Blood Pressure**. As a result, we should now see some user-generated data appear as the data file is being processed. Towards the end of execution, an error message appears.
   h. After execution of the system stops, **Stop** the trace

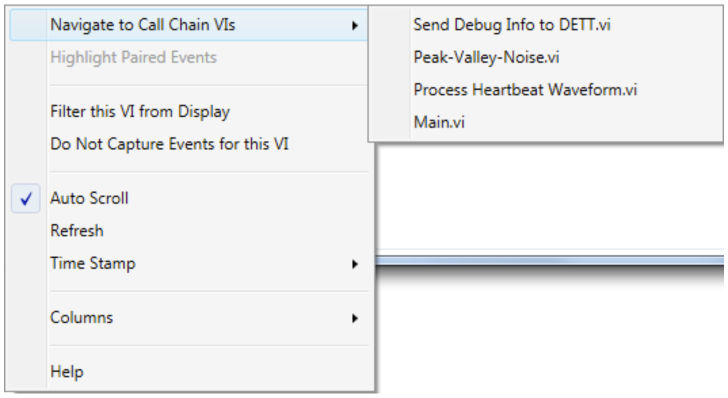
4. Understanding the Data
   a. We received several columns of data, including the VI it occurred in, the description of the event, and other information such as the timestamp
   b. Highlight an event by clicking on it, such as a memory allocation. In addition to the information in the table, the bottom of the screen is populated with additional information such as the call chain.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 12 | 11:11:12.7015838 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x1585E844; Size: 137 |
| 13 | 11:11:12.7015895 | Main.vi | User Defined | 5 | 0 | Object | \\psf\Home\Documents\Development\Software Engineering Tools\Sour |
| 14 | 11:11:12.7260193 | Main.vi:Instance:3 | Memory Allocate | 5 | 0 | VI | Handle: 0x0432EAA0; Size: 400004 |
| 15 | 11:11:12.7264151 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x043196B0; Size: 400004 |
| 16 | 11:11:12.7265200 | Main.vi | Memory Allocate | 5 | 0 | VI | Handle: 0x043092EC; Size: 201788 |

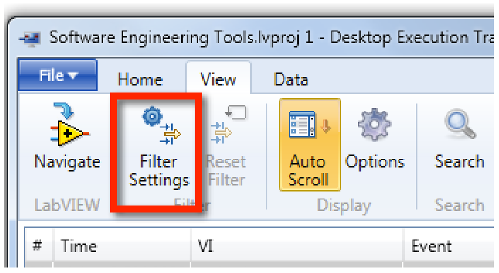
```
Details: Handle: 0x043092EC; Size: 201788
Call Chain:
-- Main.vi
VI Path: \\psf\Home\Documents\Development\Software Engineering Tools\Source\Software Engineering\Main.vi
Thread ID: 5 --- CPU ID: 0
```

c. Right-click on a row to see additional menu options, including the ability to navigate directly to any of the VIs in that particular event's call chain. You can also filter that VI from all recorded traces or future traces (if you re-start that particular trace).


Navigate to Call Chain VIs
Highlight Paired Events
Filter this VI from Display
Do Not Capture Events for this VI
Auto Scroll
Refresh
Time Stamp
Columns
Help
Send Debug Info to DETT.vi
Peak-Valley-Noise.vi
Process Heartbeat Waveform.vi
Main.vi

d. Double clicking on a row in the table will take you to the location in code that generated the captured event.

5. Using filters to parse data
   a. Click the **View** tab in the ribbon
   b. Click on **Filter Settings**



   a. Several events were captured that indicated potential problems with our application. Deselect all events, and click the boxes next to the following. Try selecting these individually to narrow the trace data further.
      i. **Errors**: two errors should be captured as a result of invalid object references
      ii. **Memory**: the Logging Slope.vi will be incrementally consuming more memory, resulting in a potential leak.
      iii. **User-Defined Trace Events**: useful debugging information that the programmer has opted to capture in order to better understand and debug the behavior of their system

6. **Optional**: The 2013 version of the Desktop Execution Trace Toolkit offers the ability to compare traces. Attempt to correct either the memory leak and/or the un-handled error and re-run the trace. Once you have multiple traces, return to the **View** tab and click **Compare Traces.** Follow the instructions in this dialog to compare the results of the two different versions of code. This is useful towards ensuring that you didn't incidentally change any other aspects of your application beyond what you intended to modify.

## TESTING AND VALIDATION

The idea behind unit testing is elegant and simple, but can be expanded to enable sophisticated series of tests for code validation and regression testing. A unit test is strictly something that 'exercises' or runs the code under test. Many developers manually perform unit testing on a regular basis in the course of working on a segment of code. In other words, it can be as simple as, 'I know the code should perform this task when I supply this input; I'll try it and see what happens.' If it doesn't behave as expected, the developer would likely modify the code and repeat this iterative process until it works.

The problem with doing this manually is that it can easily overlook large ranges of values or different combinations of inputs and it offers no insight into how much of the code was actually executed during testing. Additionally, it does not help us with the important task of proving to someone else that it worked and that it worked correctly. The cost and time required is compounded by the reality that one round of testing is rarely enough; besides fixing bugs, any changes that are made to code later in the development process may require additional investment of time and resources to ensure it's working properly.

Large projects typically augment manual procedures with tools such as the NI LabVIEW Unit Test Framework Toolkit to automate and improve this process. Automation reduces the risk of undetected errors, saves costs by detecting problems early in the development lifecycle, and saves time by keeping developers focused on the task of writing the software, instead of performing the tests themselves.

# Exercise 7: Unit Testing and Validation of Code

## Goal

We want to automate the process of testing VIs in order to make sure they exhibit correct behavior.

## Scenario

Consider that you've been given requirements for implementing a subroutine and you want to make sure it works as expected. Automating the tests makes it possible to re-run them on a regular basis and thereby mitigate the risk of making a change that could introduce a problem. We can also generate reports and get additional information about our code that can help further improve the quality and reliability of the application.

## Description

We're going to use the NI LabVIEW Unit Test Framework Toolkit to generate a test case for a simple VI, examine the results, and generate a report.

## Concepts Covered

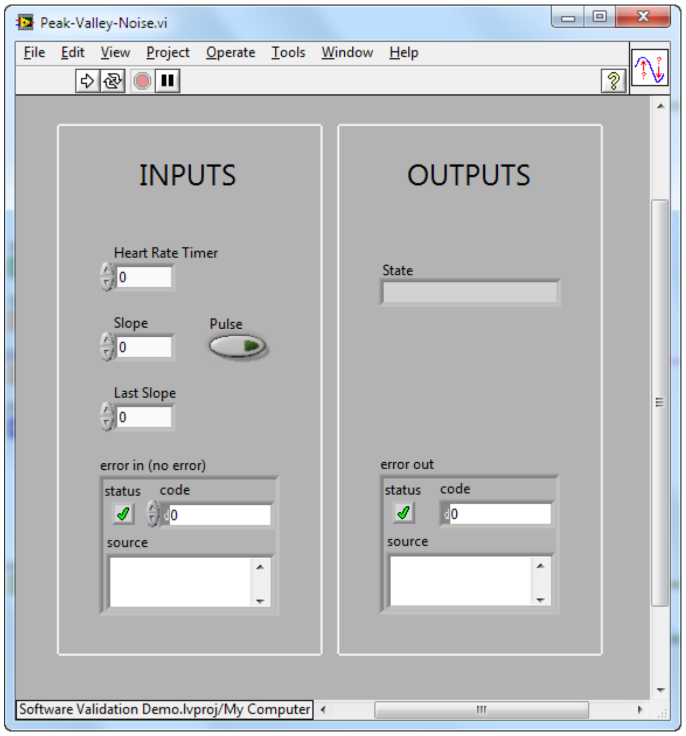
- Creating a unit test
- Defining test cases
- Tracking tests in the Project Explorer
- Importing test parameters from the front panel
- Executing tests
- Interpreting the test results dialog
- Report generation

## Setup

- Make sure the UTF Directory is setup properly
    - Right click on the project file in the Project Explorer and select properties.
    - Select 'Unit Test Framework'
    - Scroll down to 'Test Creation' and make sure that the correct directory is selected

## INSTRUCTIONS

1. **Optional:** Perform a Manual Test
   a. In the Project Explorer, expand 'Algorithms' and open 'Peak-Valley-Noise.vi.' This VI executes in a loop to identify the peaks and valleys based by comparing the instantaneous slope with the previous slope.
   b. Review the requirements document to get the test vector. Expand 'Requirements Documents' and open 'Blood Pressure Requirements – Software Unit Tests.doc'
   c. Peak-Valley-Noise.vi has the following inputs:
      i. 'Slope' – this is the rate at which the pressure is changing between the two most recent points
      ii. 'Last Slope' – this is the rate at which the pressure is changing between the previous two points
      iii. 'Heart Rate Timer'– this corresponds to the number of data-points that have been analyzed since the previous valley. If the number of samples is less than 300, it should return 'Noise'
      iv. 'Pulse' – set to True after a Valley and until a Peak
      v. 'error in (no error)' – if an error is passed into this VI, it should return the same error and zeros for both indicators.
   b. 'Peak-Valley-Noise.vi' has the following outputs:
      i. 'State' – the options are Peak, Valley or Noise
      ii. 'error out' – if invalid inputs are received by this VI, it will output an error. It will also pass any errors that are input to it.



   d. Configure the VI for a manual test
      i. Set **Heart Rate Timer** to '500'
      ii. Set **Slope** to '4

iii. Set **Last Slope** to '-9'

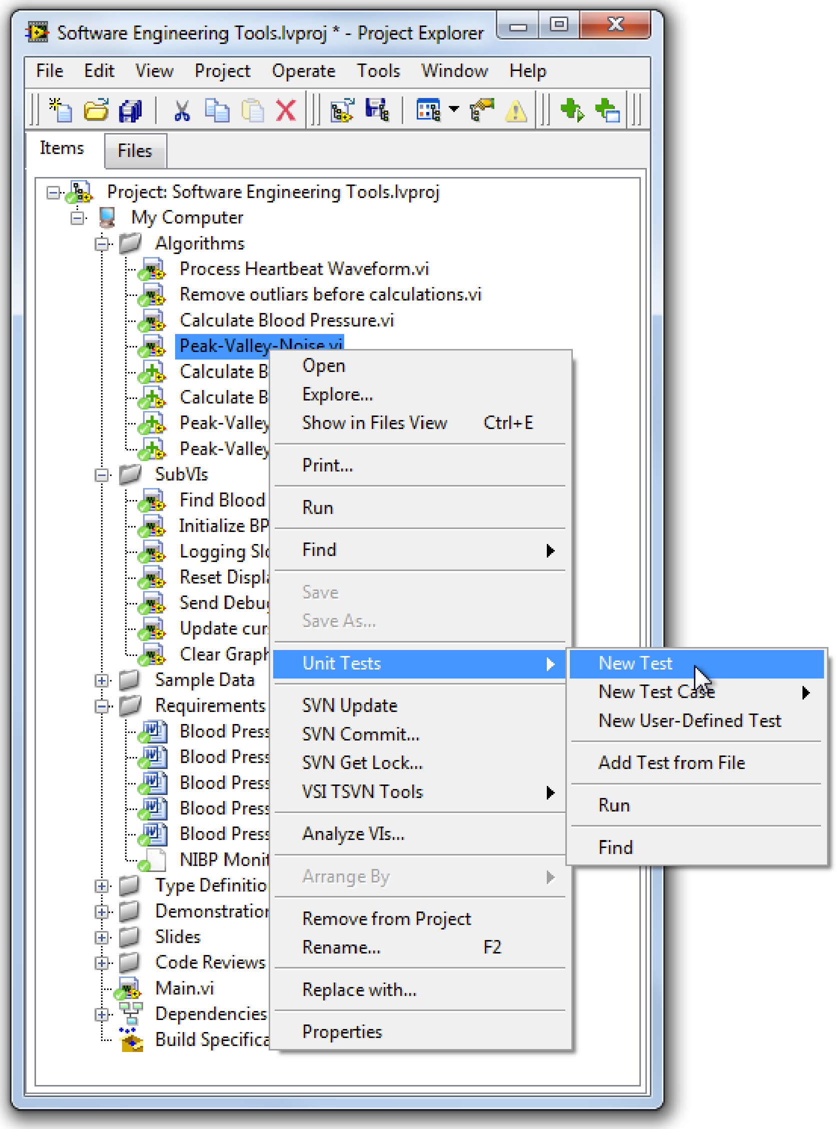
b. Run the VI. The outputs should be:

i. **State** should be 'Valley'

ii. **Error Out** status should not indicate an error

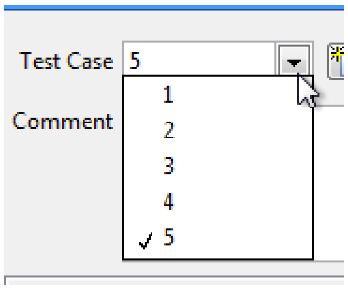
2. Define a Unit Test for **Peak-Valley-Noise.vi**

e. Though we've manually run the VI to check and make sure it works, there are numerous conditions that we want to make sure this VI can properly handle. Several tests have already been created for this VI, but we're going to start by creating a new test for positive values. Right-click on the VI and select '**Unit Tests > New Test**.'



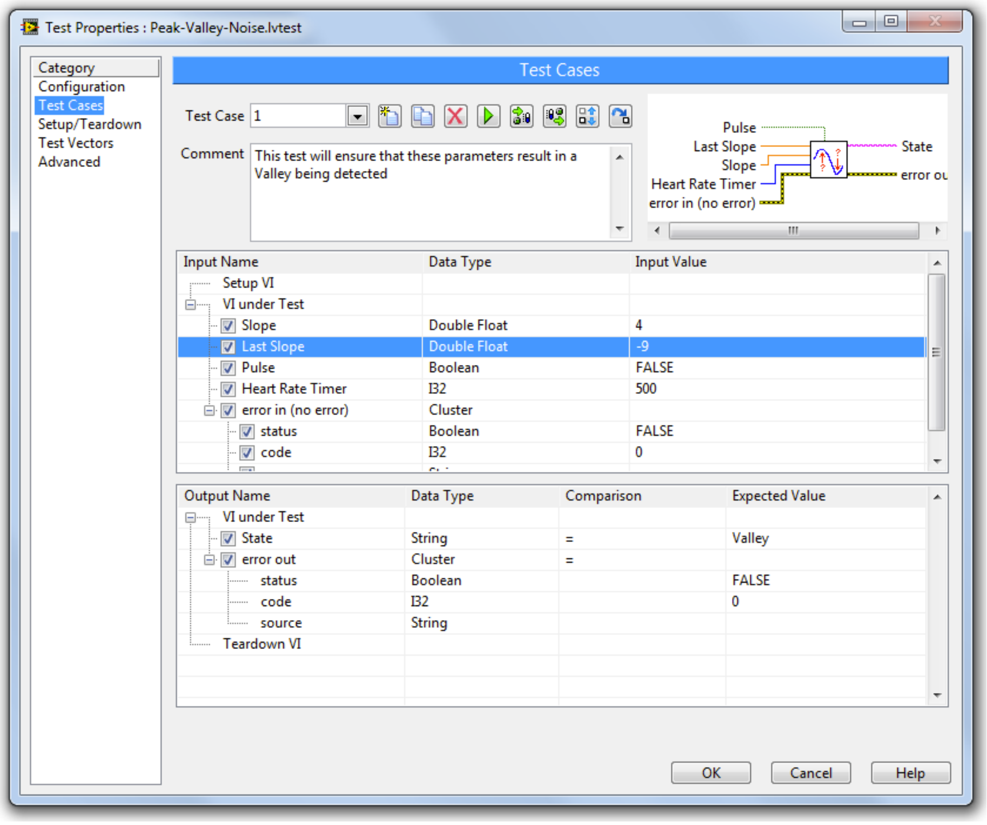
a. LabVIEW will generate a new file on disk with an .lvtest extension. The test will be created next to the VI under test by default, though we can specify a separate location or move the test on disk from within the 'Files' tab. Double-click on the unit test in the Project Explorer to open the Test Properties dialog.

b. Select the 'Test Cases' category. Note that one unit test can contain multiple test cases.



c. The right side of the 'Test Cases' dialog will display the inputs and outputs of the VI Under Test. From this dialog we can configure the following:
   i. The inputs to set
   ii. The input values
   iii. The excepted outputs
   iv. The outputs to compare
   v. The comparisons to be made between the actual results and the expected results

d. Only the controls and indicators connected to the connector pane will be shown in the Test Case dialog by default, but we can adjust the settings from the **Advanced** category to use any and all controls and indicators. Tests can be created for any data-type in LabVIEW, including arrays and clusters. For complex datasets, it may be better to define values in the .lvtest file, via the front panel or a setup VI.



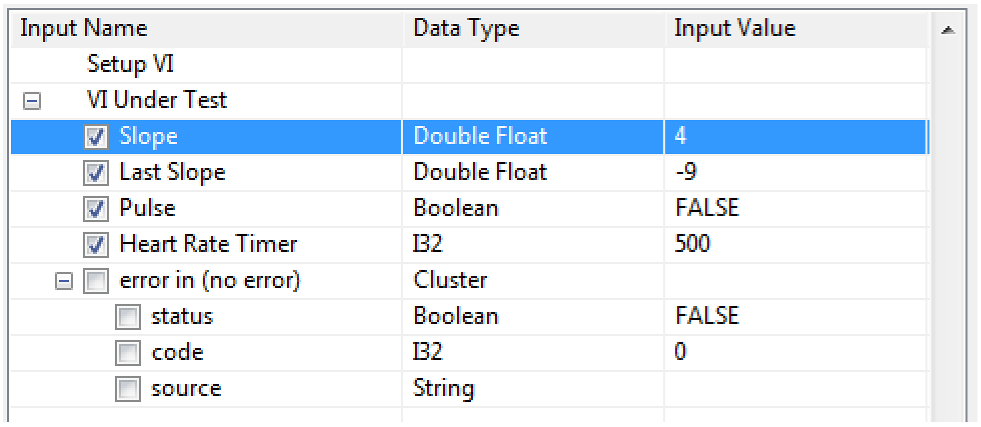
Test Case: 1

Comment: This test will ensure that these parameters result in a Valley being detected

| Input Name | Data Type | Input Value |
|---|---|---|
| Setup VI | | |
| VI under Test | | |
| ☑ Slope | Double Float | 4 |
| ☑ Last Slope | Double Float | -9 |
| ☑ Pulse | Boolean | FALSE |
| ☑ Heart Rate Timer | I32 | 500 |
| ☑ error in (no error) | Cluster | |
| ☑ status | Boolean | FALSE |
| ☑ code | I32 | 0 |

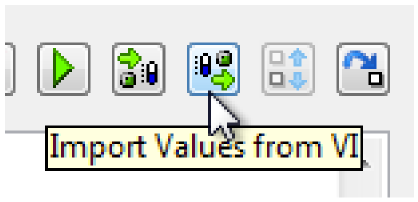
| Output Name | Data Type | Comparison | Expected Value |
|---|---|---|---|
| VI under Test | | | |
| ☑ State | String | = | Valley |
| ☑ error out | Cluster | = | |
| status | Boolean | | FALSE |
| code | I32 | | 0 |
| source | String | | |
| Teardown VI | | | |

e. Define the input values for the values based on the requirements document opened in an earlier step. Alternatively, you can also just copy the values below:



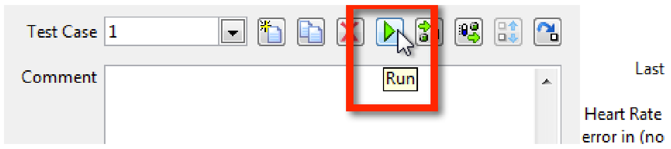
| Input Name | Data Type | Input Value |
|---|---|---|
| Setup VI | | |
| ⊟ VI Under Test | | |
| ☑ Slope | Double Float | 4 |
| ☑ Last Slope | Double Float | -9 |
| ☑ Pulse | Boolean | FALSE |
| ☑ Heart Rate Timer | I32 | 500 |
| ⊟ ☐ error in (no error) | Cluster | |
| ☐ status | Boolean | FALSE |
| ☐ code | I32 | 0 |
| ☐ source | String | |

f. **Optional:** If you completed the first step in this exercise to manually test the VI and you left the Front Panel open, you can import the values from the Front Panel by clicking **Import Values from VI**

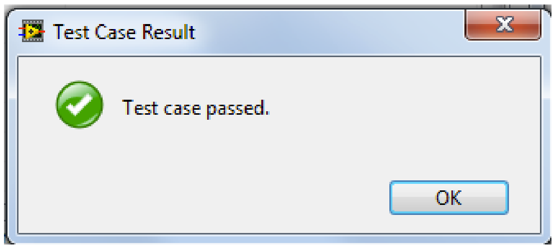


a. **Optional:** Test parameters can also be modified outside of LabVIEW using tools like Excel. If Excel is installed, you can right-click on the unit test from within the Project Explorer and select **Open in external editor**

b. Define the output string as 'Valley'
c. Feel free to define additional test cases from this dialog by clicking **New** at the top.
d. As of LabVIEW 2013, tests can be run directly from the editor. Without closing the dialog, click the run arrow in the menu bar to execute this test.

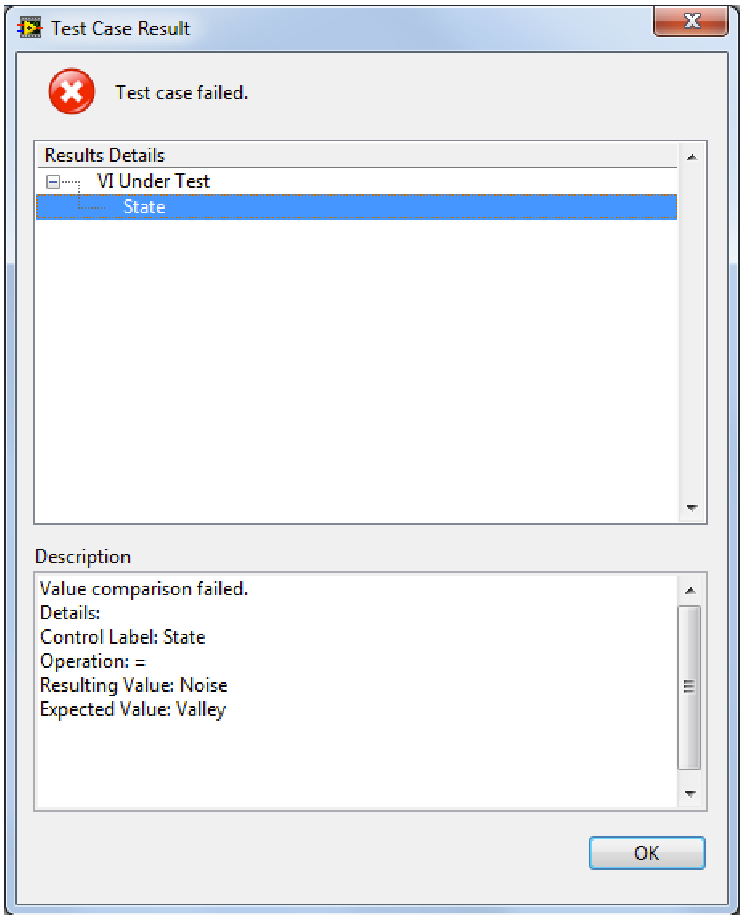


e. When completed, if the VI works correctly, you should see that the test passed


Test Case Result

Test case passed.

f. If the test does not pass, an failure dialog will appear that explains which parameters did not return expected values:


Test Case Result

Test case failed.

Results Details
VI Under Test
State

Description
Value comparison failed.
Details:
Control Label: State
Operation: =
Resulting Value: Noise
Expected Value: Valley

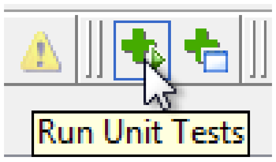
g. The Unit Test Framework is also designed to enable rapid iteration on test parameters and the code to address any failures. With the UTF dialog still open, the VI can be modified and run in order to iterate and quickly address any problems. Front panel control values can also be imported and exported to a specific test case.

h. **Optional**: Click on the 'Configuration' category, which shows the basic configuration of the unit test. The information displayed includes the following:
    i. **VI Under Test** – this will automatically be configured, but we can change it at a later date if we move or rename a VI under test outside of the LabVIEW Project Explorer.
    ii. **Test Priority** – this number can be used to group tests and test results based upon importance. As an example, you can tell the Unit Test Framework to only run tests that are at least a certain priority.
    iii. **Requirements ID (Optional)** – this ID can be read by NI Requirements Gateway for the sake of automated traceability to requirements documents.

b. **Optional:** On the 'Configuration' category page, enter the requirement ID: SwTestReq1 into the **RequirementsID** field. This can be parsed by NI Requirements Gateway

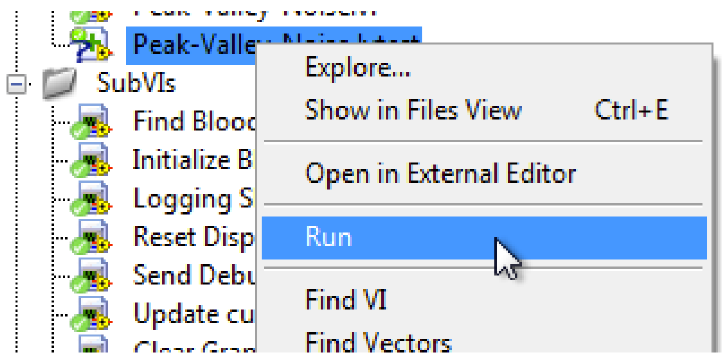
i. Click **OK**

j. Now that the Unit Test Framework editor has been closed, tests can be run several different ways:
    i. They can be run individually by right-clicking on them and selecting **Run**
    ii. There is a toolbar item in the project to run all tests

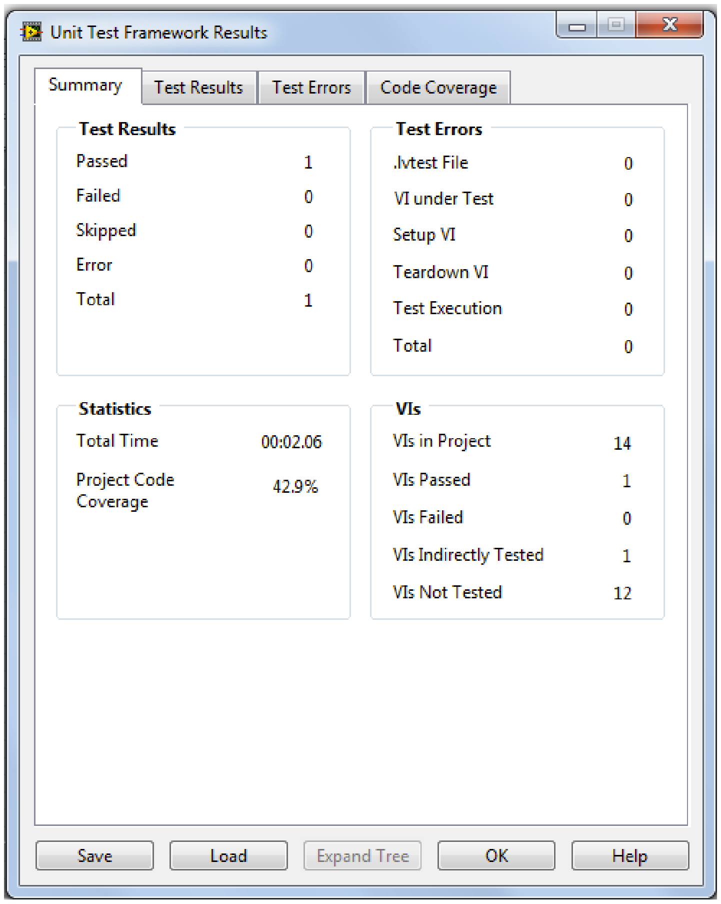


    iii. Folders containing unit tests have a right-click menu item that allows the execution of all tests within
    iv. The UTF API allows developers to execute these tests programmatically.

k. Right click on the test in the Project Explorer and select **Run**



m. The test should pass, which will be indicated by a green icon that is overlaid on the test in the Project tree. A dialog will also appear explaining the results.



**Test Results**

| | |
|---|---|
| Passed | 1 |
| Failed | 0 |
| Skipped | 0 |
| Error | 0 |
| Total | 1 |

**Test Errors**

| | |
|---|---|
| .lvtest File | 0 |
| VI under Test | 0 |
| Setup VI | 0 |
| Teardown VI | 0 |
| Test Execution | 0 |
| Total | 0 |

**Statistics**

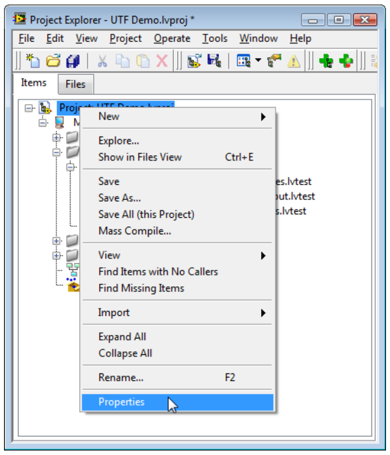
| | |
|---|---|
| Total Time | 00:02.06 |
| Project Code Coverage | 42.9% |

**VIs**

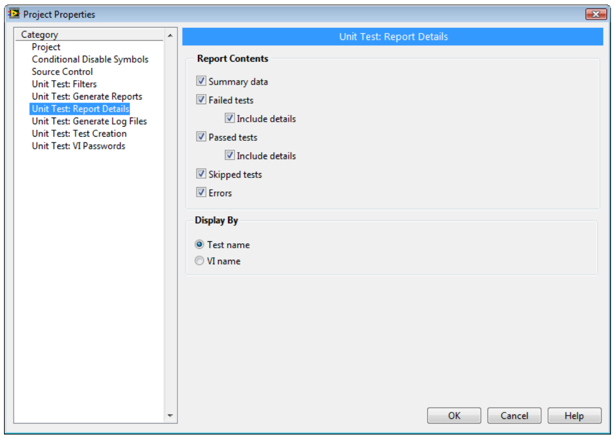
| | |
|---|---|
| VIs in Project | 14 |
| VIs Passed | 1 |
| VIs Failed | 0 |
| VIs Indirectly Tested | 1 |
| VIs Not Tested | 12 |

n. Click **OK**

3. Turn on Report Generation
    a. In the Project Explorer, right click on the project file and select properties



    b. The properties dialog contains various settings and preferences for the Unit Test Framework, including test filters and the default location for new tests
    c. Select Unit Test: Report Details and check everything



Unit Test: Report Details

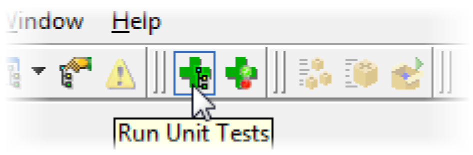
Report Contents

- [✓] Summary data
- [✓] Failed tests
  - [✓] Include details
- [✓] Passed tests
  - [✓] Include details
- [✓] Skipped tests
- [✓] Errors

Display By

- [✓ Test name]
- VI name

    d. Select Unit Test: Generate Reports and select Generate HTML Report and View Report after Execution
    e. Click on the icon in the toolbar to Run Unit Tests

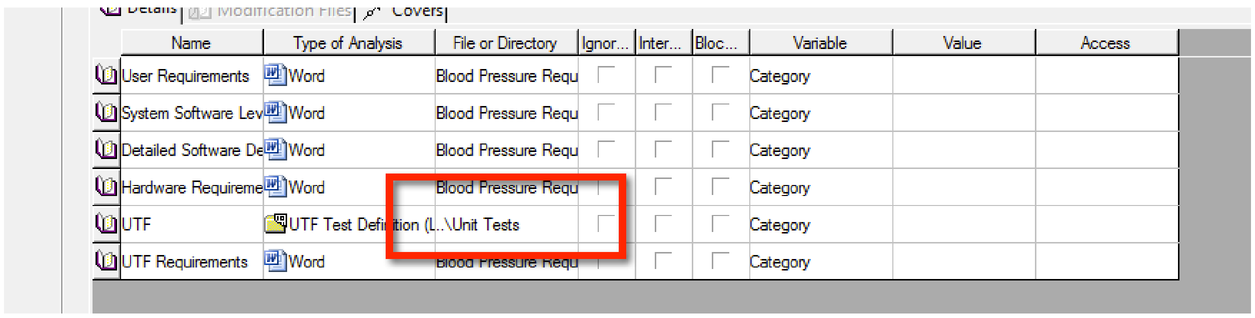


    f. After the tests are complete, the HTML report should display in your browser.
4. **Optional:** View the traceability of the newly created test files in NI Requirements Gateway. This will only show coverage if you completed the earlier optional step to add the requirements ID to the lvtest file.
    a. Move all of the unit test files into a unique folder on disk, as we will point Requirements Gateway to this location

b. A requirements document has already been included for the unit tests, 'Book Pressure Requirements – Software Unit Tests.doc.' Return to the same Requirements Gateway project used in Exercise 4 and open the configuration dialog by clicking **File > Edit Project**.

c. A line item has already been included for the unit tests. Point this item to the directory containing the lvtest files.

d. Click **OK** and allow NI Requirements Gateway to re-parse everything.


Details | Modification Files | Covers

| Name | Type of Analysis | File or Directory | Ignor... | Inter... | Bloc... | Variable | Value | Access |
|---|---|---|---|---|---|---|---|---|
| User Requirements | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| System Software Lev | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| Detailed Software De | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| Hardware Requireme | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |
| UTF | UTF Test Definition (L | ..\Unit Tests | ☐ | ☐ | ☐ | Category | | |
| UTF Requirements | Word | Blood Pressure Requ | ☐ | ☐ | ☐ | Category | | |

# More Information

## Download Source Code, Manual and Slides

- **http://bit.ly/lv_swe**

## Online Resources

- **ni.com/largeapps** – find best practices, online examples and a community of advanced LabVIEW users
- **ni.com/community/largeapps** – participate in a community of other LabVIEW users developing large applications
- **ekerry.wordpress.com** – follow Elijah Kerry's blog on software engineering and large application development with LabVIEW

## Customer Education Classes

- Managing Software Engineering with LabVIEW
    - Learn to manage the development of a LabVIEW project from definition to deployment
    - Select and use appropriate tools and techniques to manage the development of a LabVIEW application
    - Recommended preparation for Certified LabVIEW Architect exam
- Advanced Architectures in LabVIEW
    - Gain exposure to and experience with various architectures for medium to large LabVIEW applications
    - Learn how to select an appropriate architecture based on high-level requirements
    - Recommended preparation for Certified LabVIEW Architect exam
- Object-Oriented Design and Programming in LabVIEW
    - Design an application using object-oriented design principles
    - Implement a basic class hierarchy using LabVIEW classes
    - Modify an existing LabVIEW application to replace common patterns with LabVIEW objects