

(PRACTICAL) COMPUTATIONAL PHYSICS

Physics 551
Lecture 15

Extra Reading

Optional Exercise

Recommended

NOTATION

- This lecture slides for this course will attempt to use a uniform notation throughout. A normal paragraph looks like this.
- ☞ *Italicized paragraphs with pen bullets will indicate definitions, with the defined word or phrase shown in **SMALL-CAPS**.*
- ☞ Pencil bullets will indicate the introduction of **new notation**.
- ☞ Pointing hand bullets indicate important points that might otherwise be overlooked.

ANNOUNCEMENTS

ANNOUNCEMENTS

- To **clone** this week's **LabVIEW demonstration materials** please open the **Git Shell** utility and invoke

```
$ git clone https://github.com/hughdickinson/CompPhysL15Labview.git
C:/Users/ComputationalPhysics/Documents/labview/lecture15
```

- To **clone** this week's **C++ demonstration materials** please open the **Git Shell** utility and invoke

```
$ git clone https://github.com/hughdickinson/CompPhysL15CPP.git
C:/Users/ComputationalPhysics/Documents/cPlusPlus/lecture15
```

- ☞ You can also find this command on the Blackboard Learn website.

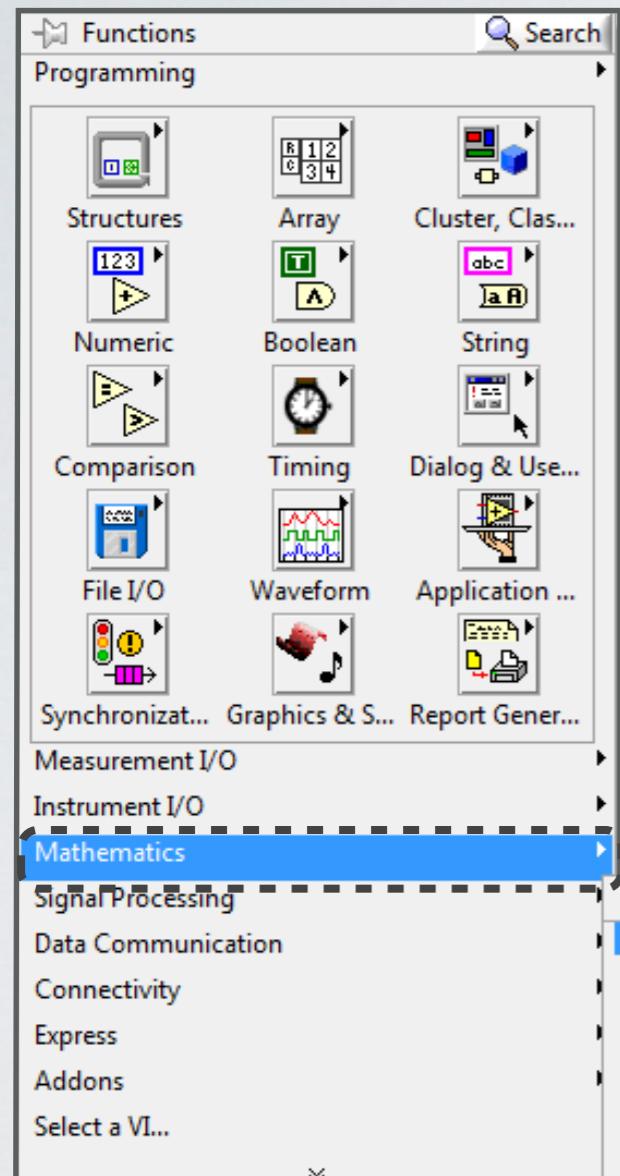
LABVIEW

SIMPLIFYING THE BLOCK DIAGRAM

FORMULA NODES

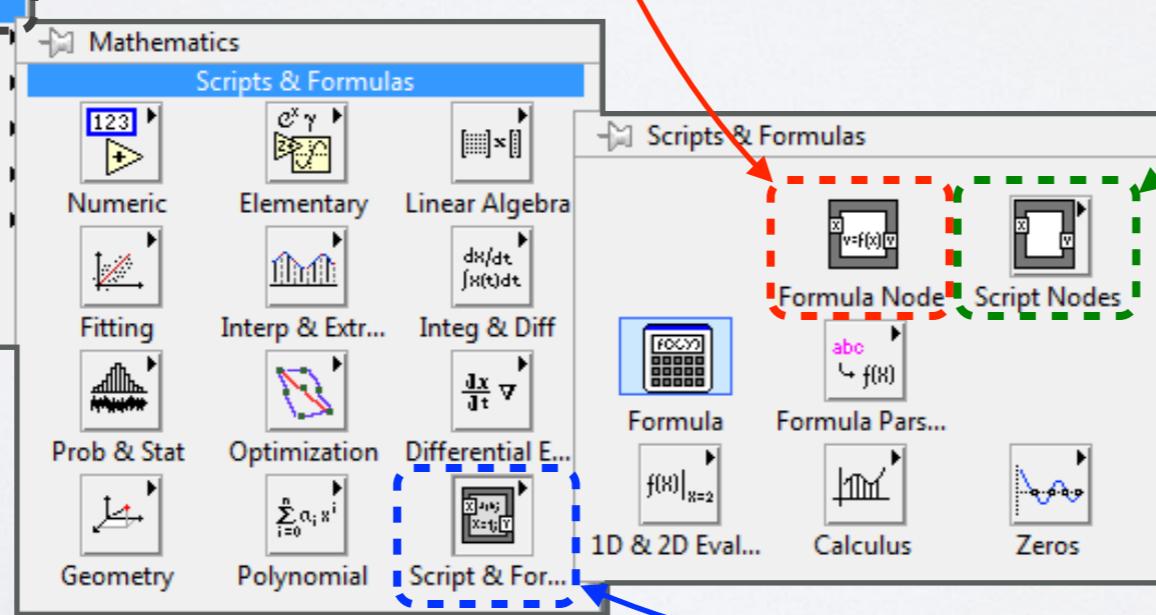
- Implementing **moderately complex** data manipulation by connecting functions from the **Numeric Palette** rapidly **complicates** the block diagram.
 - ☞ LabVIEW provides a **FORMULA NODE**, which allows complicated **data-manipulation** operations to be defined using a **C++** or **C-like syntax**.
- Formula nodes accept an **arbitrary** number of named **input** terminals and may output data through an **arbitrary** number of **output** terminals.

FORMULA NODES



Formula Nodes
Enable data manipulation using a C-style syntax

Script Nodes
Enable invocation of MATLAB scripts **if** MATLAB is available on the machine running LabVIEW

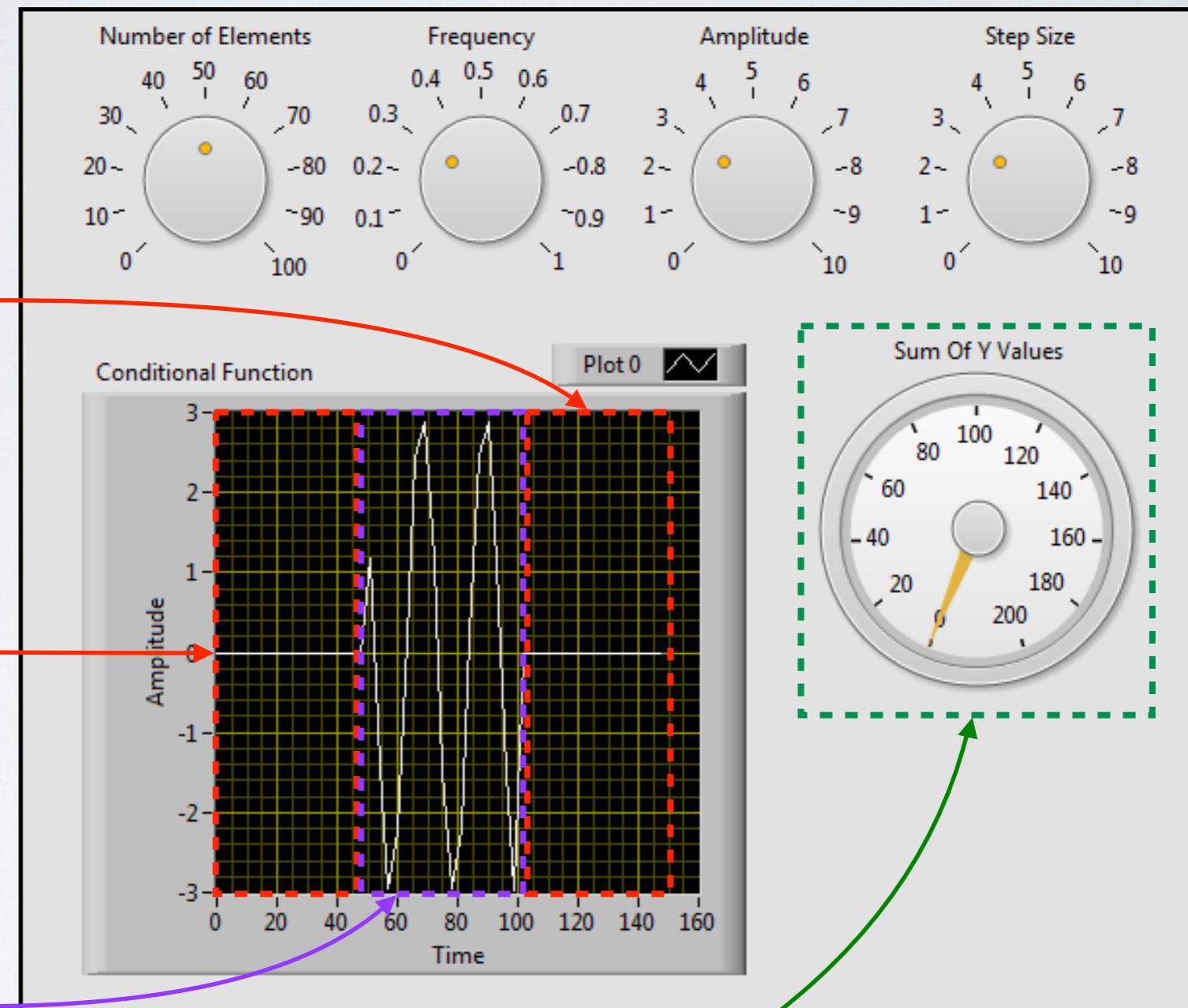


Scripts & Formulas
Palette

FORMULA NODE DETAILS

- ☞ Formula nodes cannot **generate array-type** data. Any array-type outputs must have an **identically named and dimensioned** array-type input.
- ☞ Formula nodes can substantially simplify block diagrams that implement **conditional branching** using **Case Structures**.
- ☞ Several **built-in mathematical functions** can be used within **Formula Nodes**. See the help for a full list of available functions.
- ☞ The `sizeOfDim(...)` function can be used to obtain the extent of array-type input data.

FORMULA NODE EXAMPLE



Final third
Set to zero

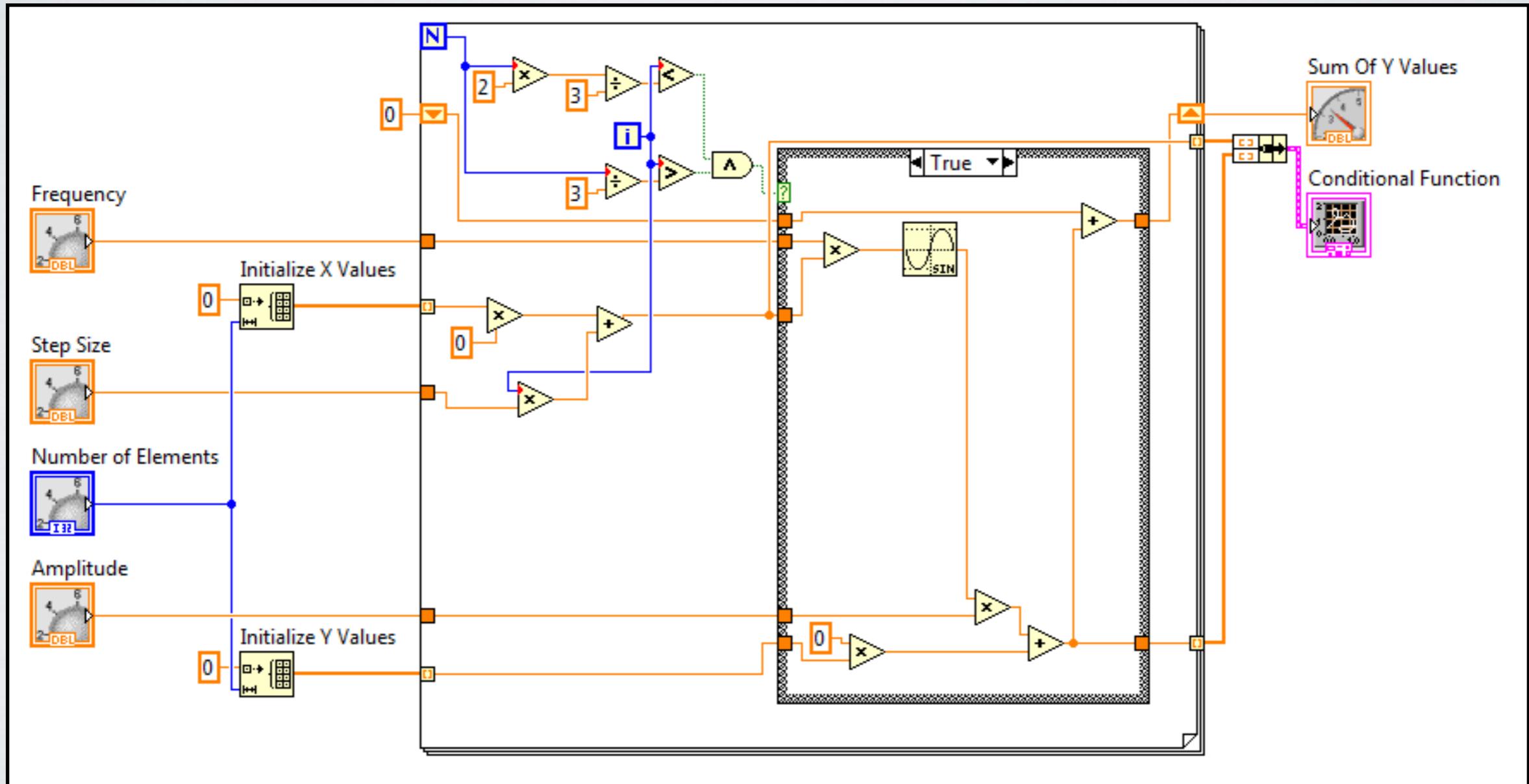
First third
Set to zero

Second third
Set to $A \cdot \sin(f \cdot x)$

Compute sum of all samples.

FORMULA NODE EXAMPLE

THE HARD WAY!

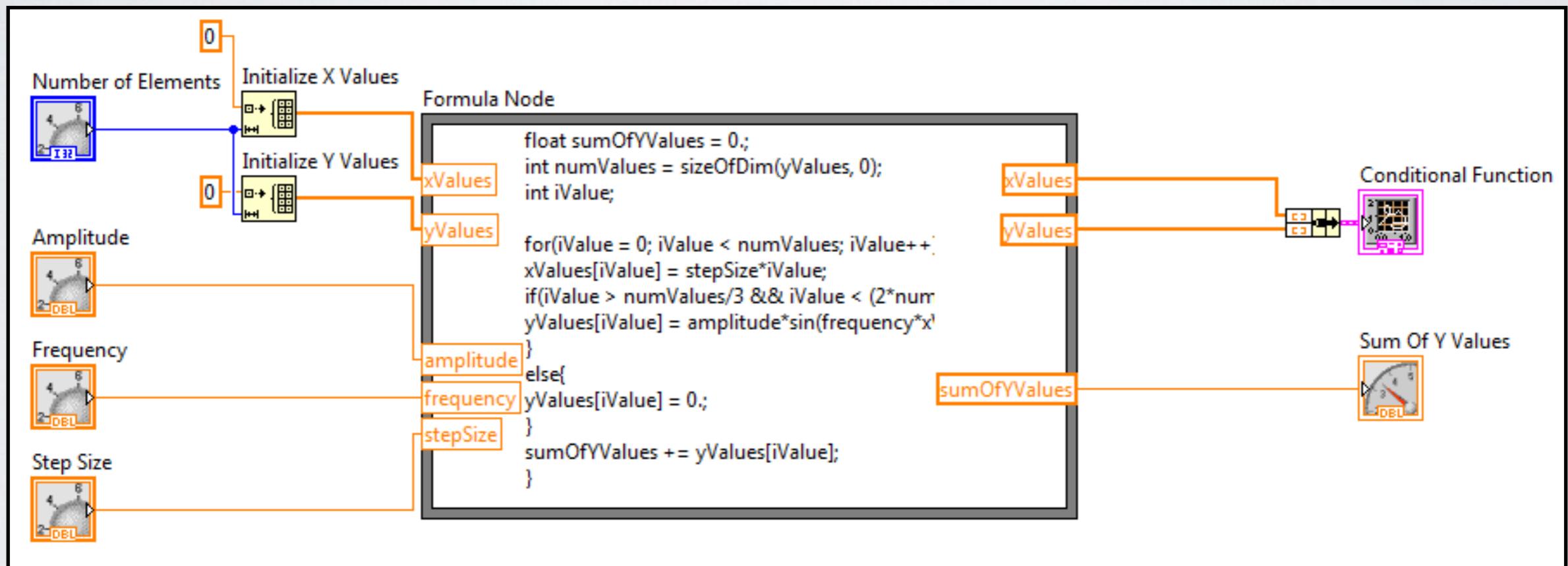


DEMONSTRATION

Using **Formula Nodes**

FORMULA NODE EXAMPLE

THE EASY WAY!

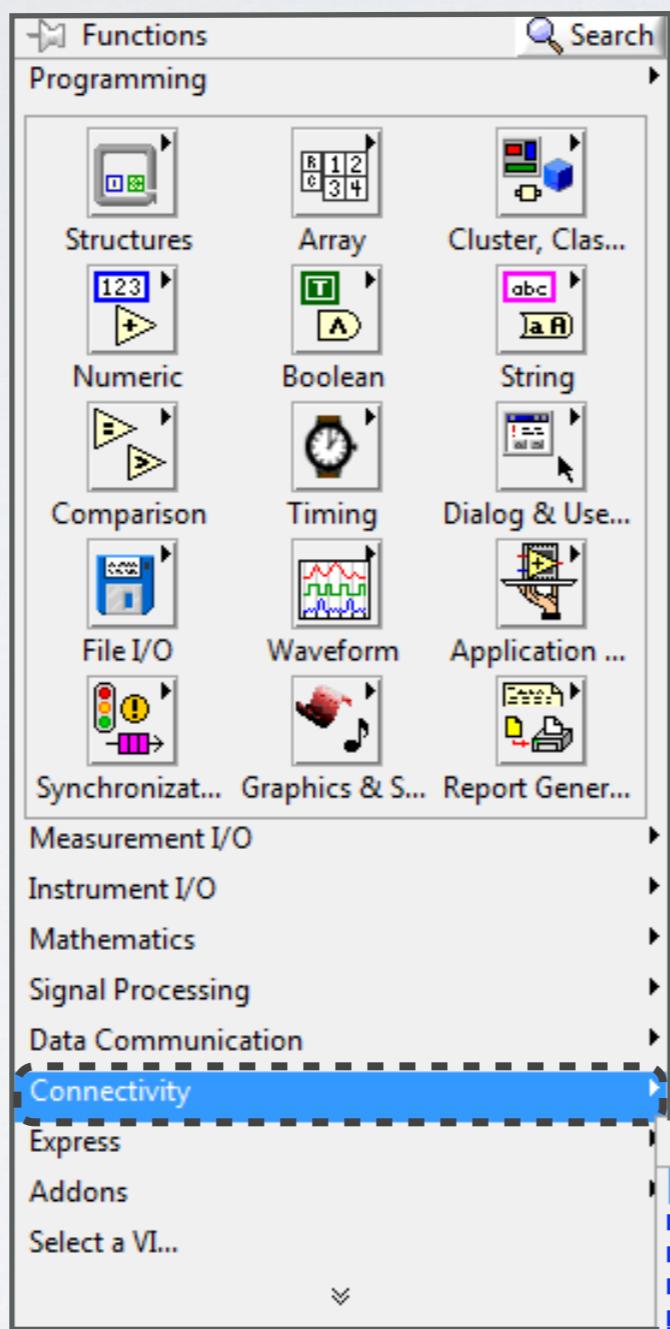


INVOKING EXTERNAL UTILITIES

AUGMENTING LABVIEW WITH EXTERNAL UTILITY

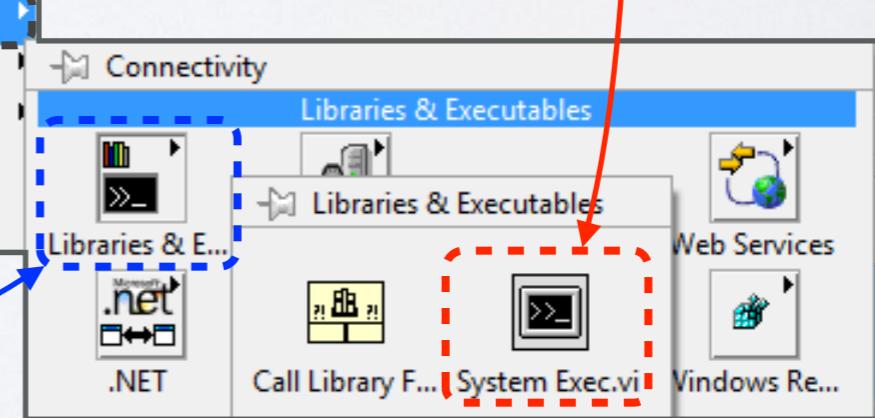
- LabVIEW provides an **extensive** framework of **built-in** data acquisition and analysis **functionality**.
 - Occasionally, highly **specific analyses** may benefit from **additional functionality** provided by external utilities e.g. Python.
- ☞ LabVIEW provides the **SYSTEM EXEC VI**, which allows an external utility to be invoked from the block diagram.
- ☞ The System Exec VI provides a lot of **flexibility** to control how the utility is invoked and how its output is handled.

THE SYSTEM EXEC VI



**Libraries &
Executables
Palette**

System Exec VI
Enables invocation of any **executable** utility that could be run from the system **command prompt** or **shell**. Command line **arguments** can also be specified and any **output** can be **captured**.



SYSTEM EXEC EXAMPLE

PYTHON INTERPRETER

| | | | | | | | | | | | | | | |
|---|--|------|--|-----|--------|--|---|--------|------|--|-----|--------|--|-------------------------------------|
| <p>Python Code</p> <pre>print "Hello, World" print [i for i in range(20) if i%2==0] this is not valid</pre> <p><i>Enter Python Code</i></p> | <p>Python Output</p> <pre>Hello, World [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]</pre> <p><i>Print Output</i></p> | | | | | | | | | | | | | |
| <p>Command line arguments</p> <p><i>Supply optional arguments to the Python Interpreter</i></p> | <p>Python Error Messages</p> <pre>Traceback (most recent call last): File "C:\Users\ComputationalPhysics\AppData\Local\Temp\lvtemporary_567977.py", line 3, in <module> this is not valid NameError: name 'this' is not defined</pre> <p><i>Print Python Errors</i></p> | | | | | | | | | | | | | |
| <p>Python Invocation Error</p> <table border="1"><tr><td>status</td><td>code</td></tr><tr><td></td><td>d 0</td></tr><tr><td>source</td><td></td></tr></table> | status | code | | d 0 | source | | <p>File Deletion Error</p> <table border="1"><tr><td>status</td><td>code</td></tr><tr><td></td><td>d 0</td></tr><tr><td>source</td><td></td></tr></table> | status | code | | d 0 | source | | <p><i>Report LabVIEW Errors</i></p> |
| status | code | | | | | | | | | | | | | |
| | d 0 | | | | | | | | | | | | | |
| source | | | | | | | | | | | | | | |
| status | code | | | | | | | | | | | | | |
| | d 0 | | | | | | | | | | | | | |
| source | | | | | | | | | | | | | | |

DEMONSTRATION

Using the **System Exec VI**

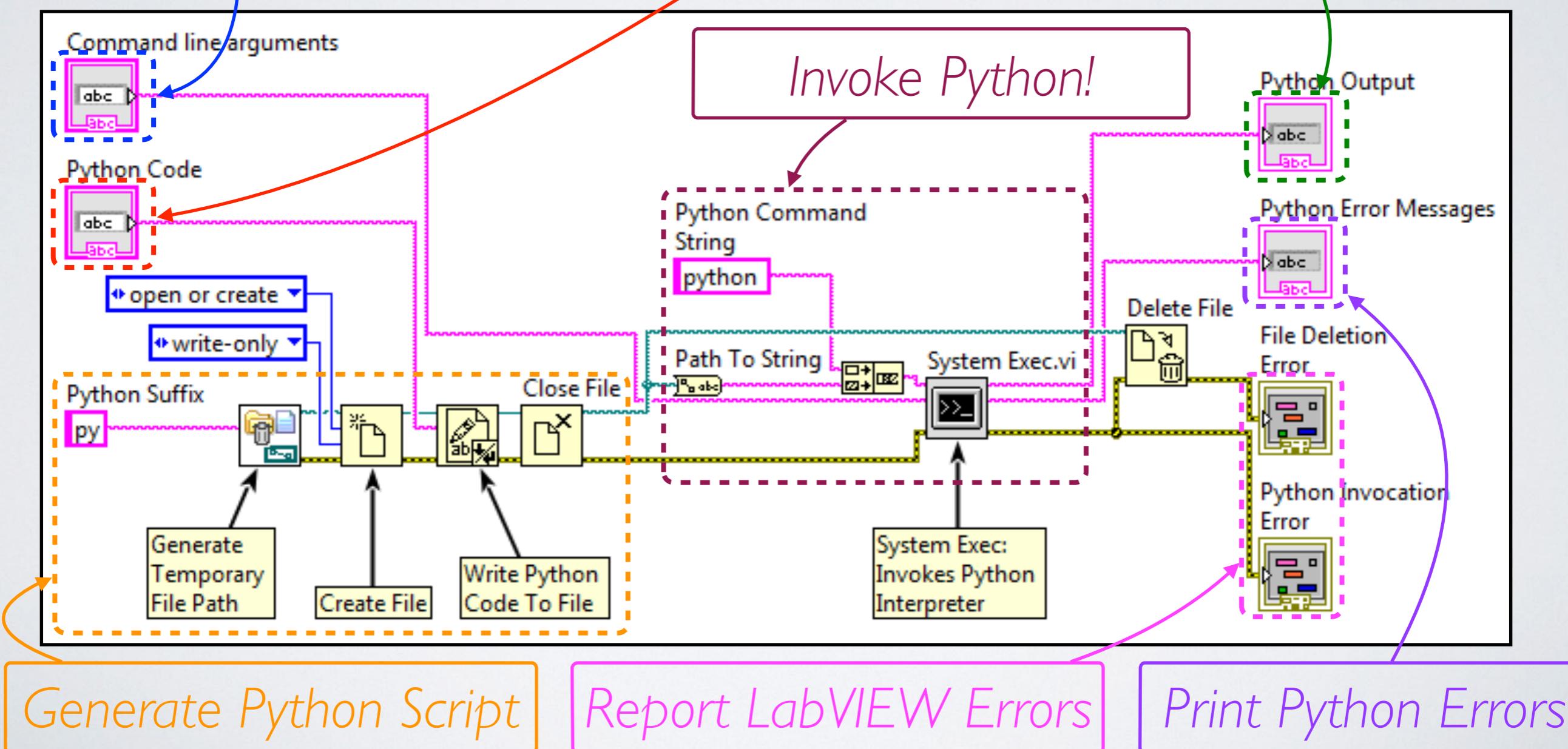
SYSTEM EXEC EXAMPLE

PYTHON INTERPRETER

Python Interpreter
Arguments

Enter Python Code

Print Output



WORKING WITH C++ SHARED LIBRARIES

REUSING LEGACY ANALYSIS CODE

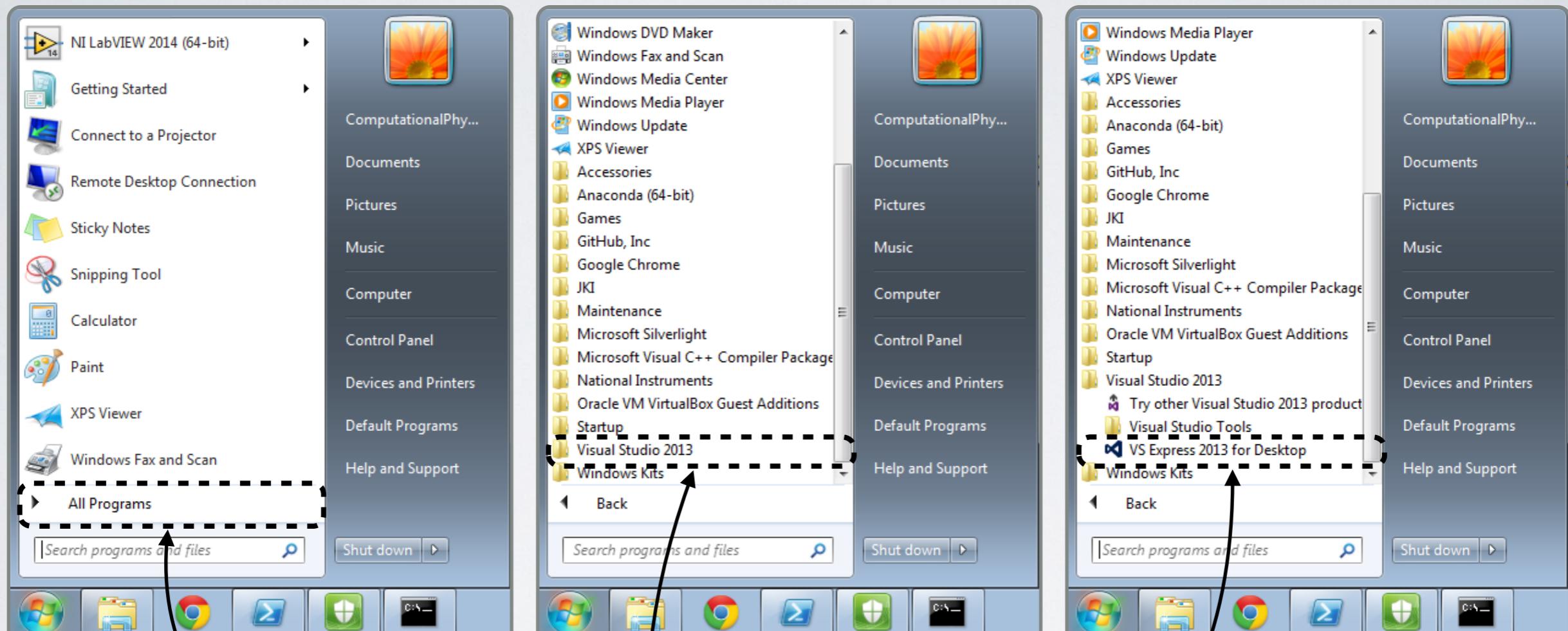
- LabVIEW is able to **import functions** provided by **shared libraries** and **generate** corresponding **VIs** that can be added to the Block Diagram.
 - ☞ LabVIEW automatically **maps** between function **parameter** and **return** types and LabVIEW data types that can be connected.
 - ☞ An **intuitive wizard** interface is provided to **simplify** the importation process, making it **straightforward** to reuse **legacy analysis routines**.
 - ☞ You may have to **build a shared library in Windows**.

DEMONSTRATION

Importing **shared library functions** as **LabVIEW VIs**

STEP I:
START VISUAL STUDIO

START VISUAL STUDIO



I) All Programs

3) VS Express 2013 for Desktop

2) Visual Studio 2013

START VISUAL STUDIO

First Time Startup

Skip Sign In



Welcome. Sign in to Visual Studio.

Visual Studio will automatically keep you signed in, sync your settings between devices, and connect to online developer services.

[Learn more](#)

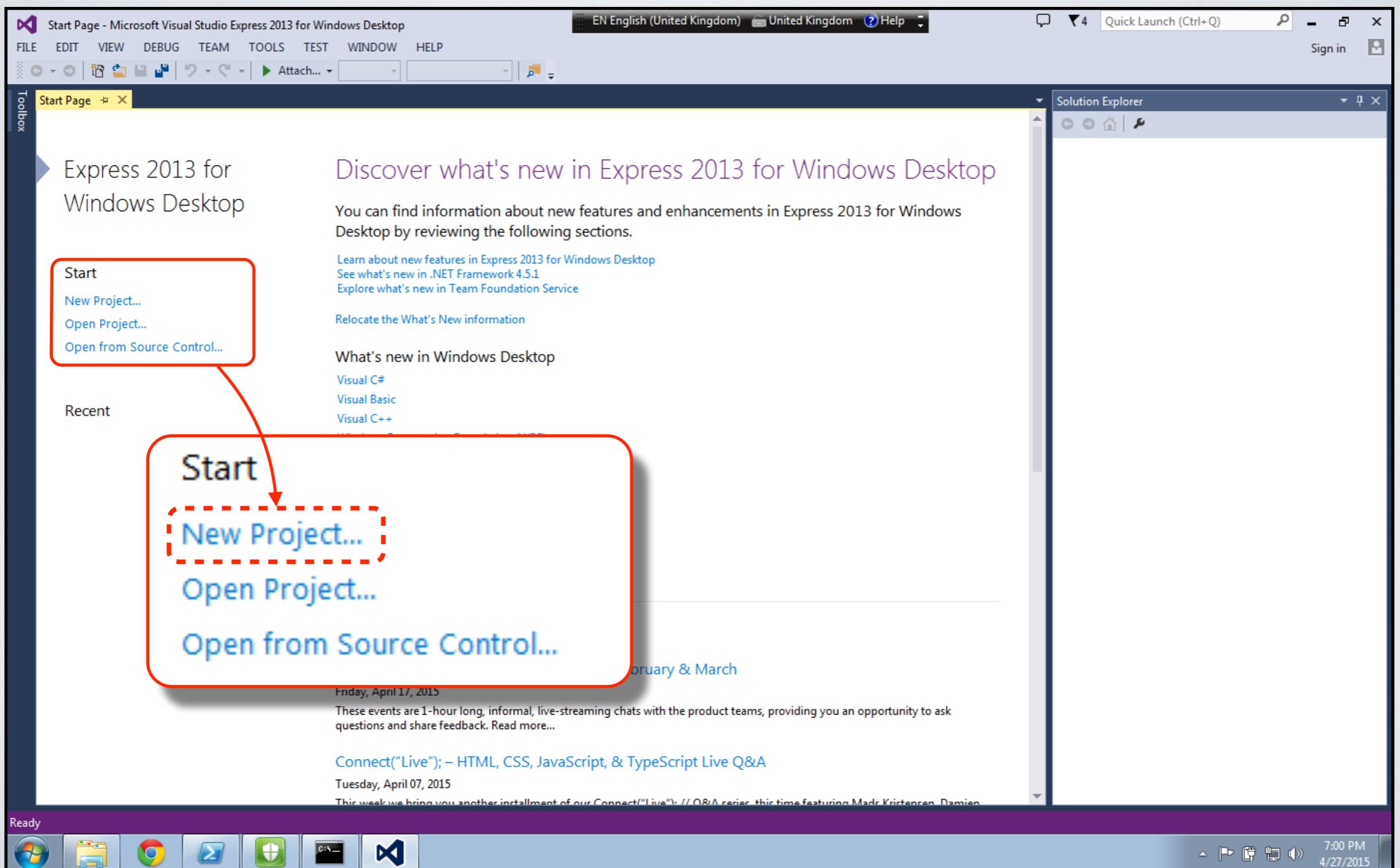
[Sign in](#)

[Not now, maybe later.](#)

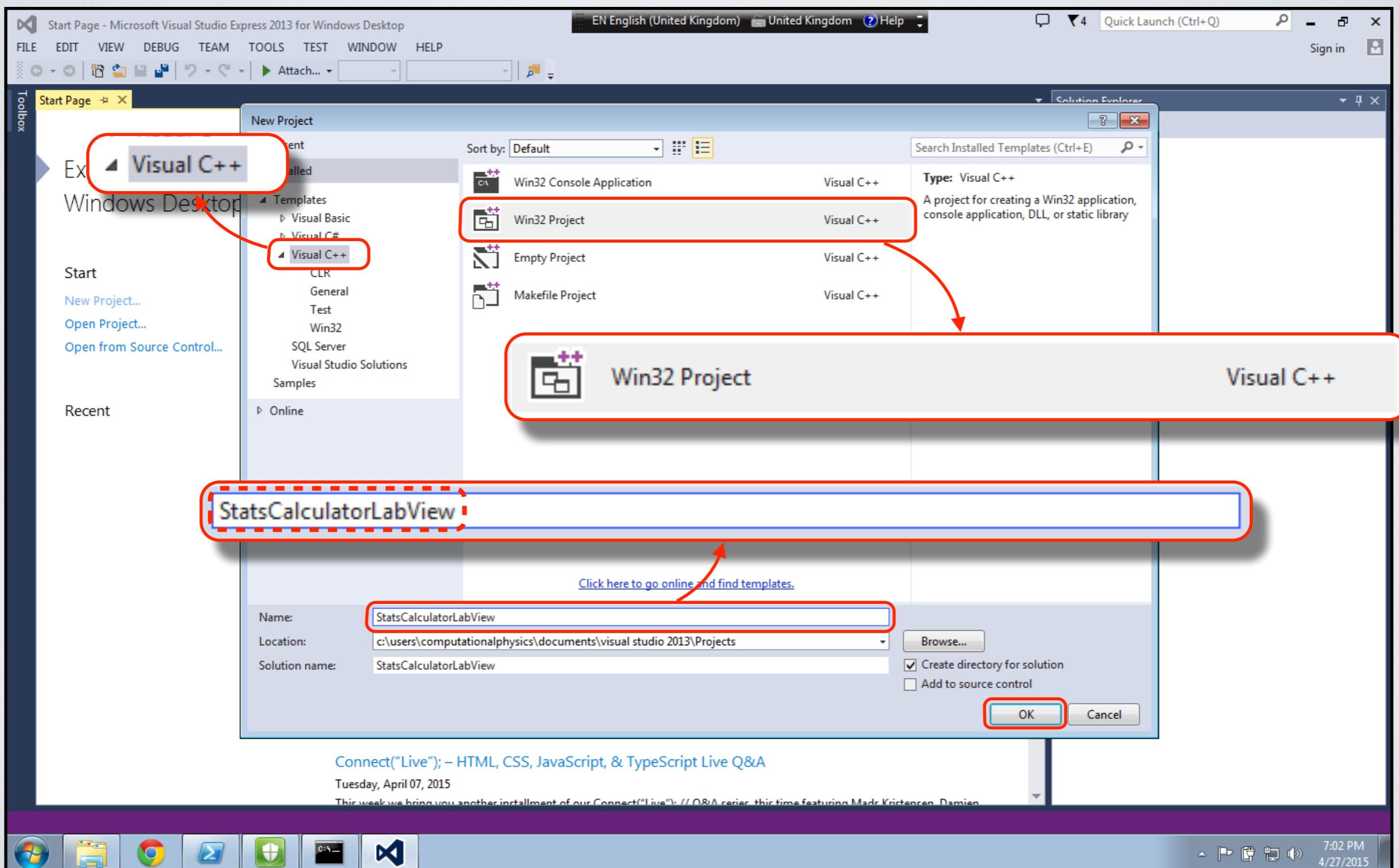
By signing in, you agree to the Team Foundation Services
[Terms of Use](#) and [Privacy Statement](#)

STEP 2:
CREATE A NEW PROJECT

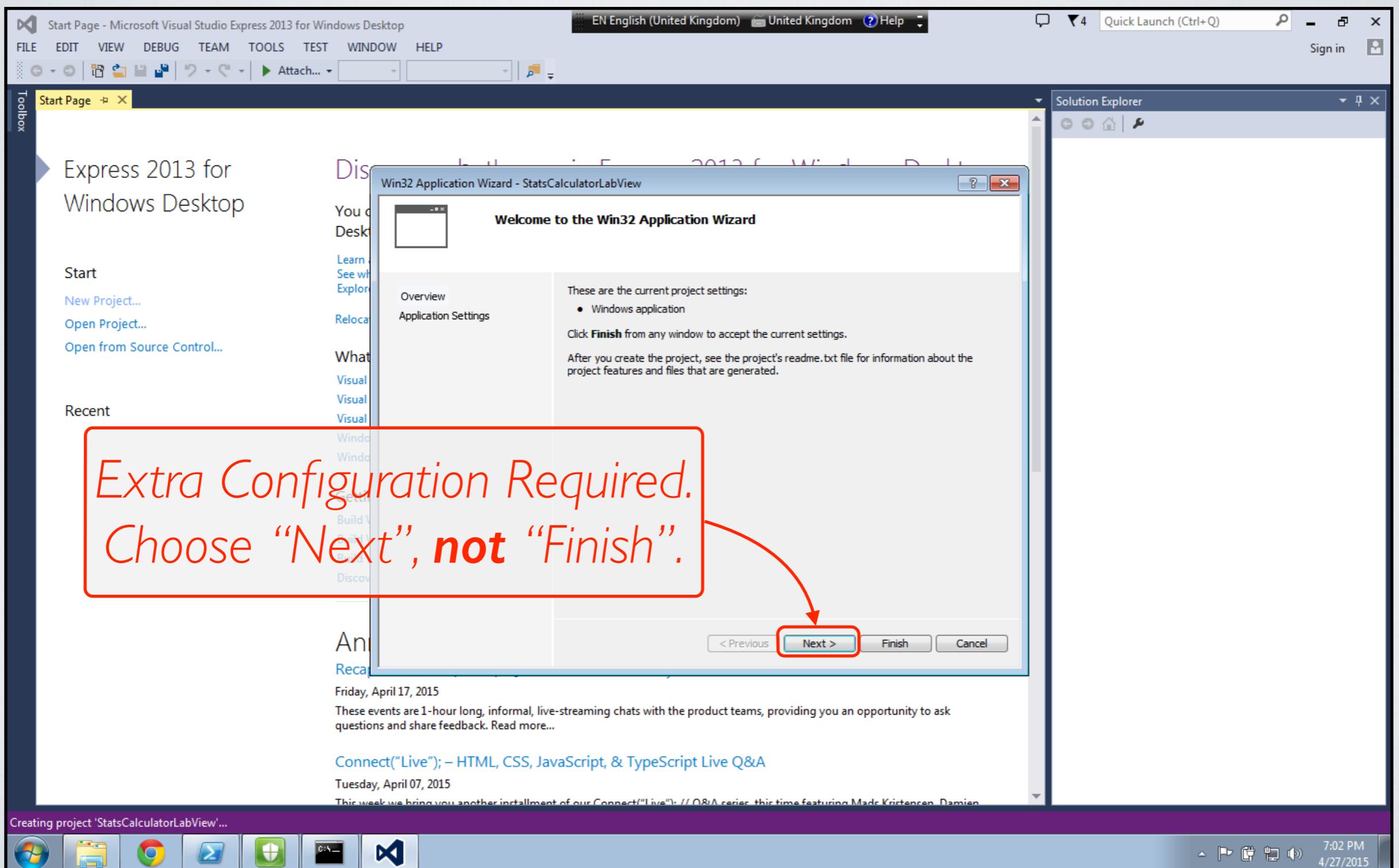
CREATE NEW PROJECT



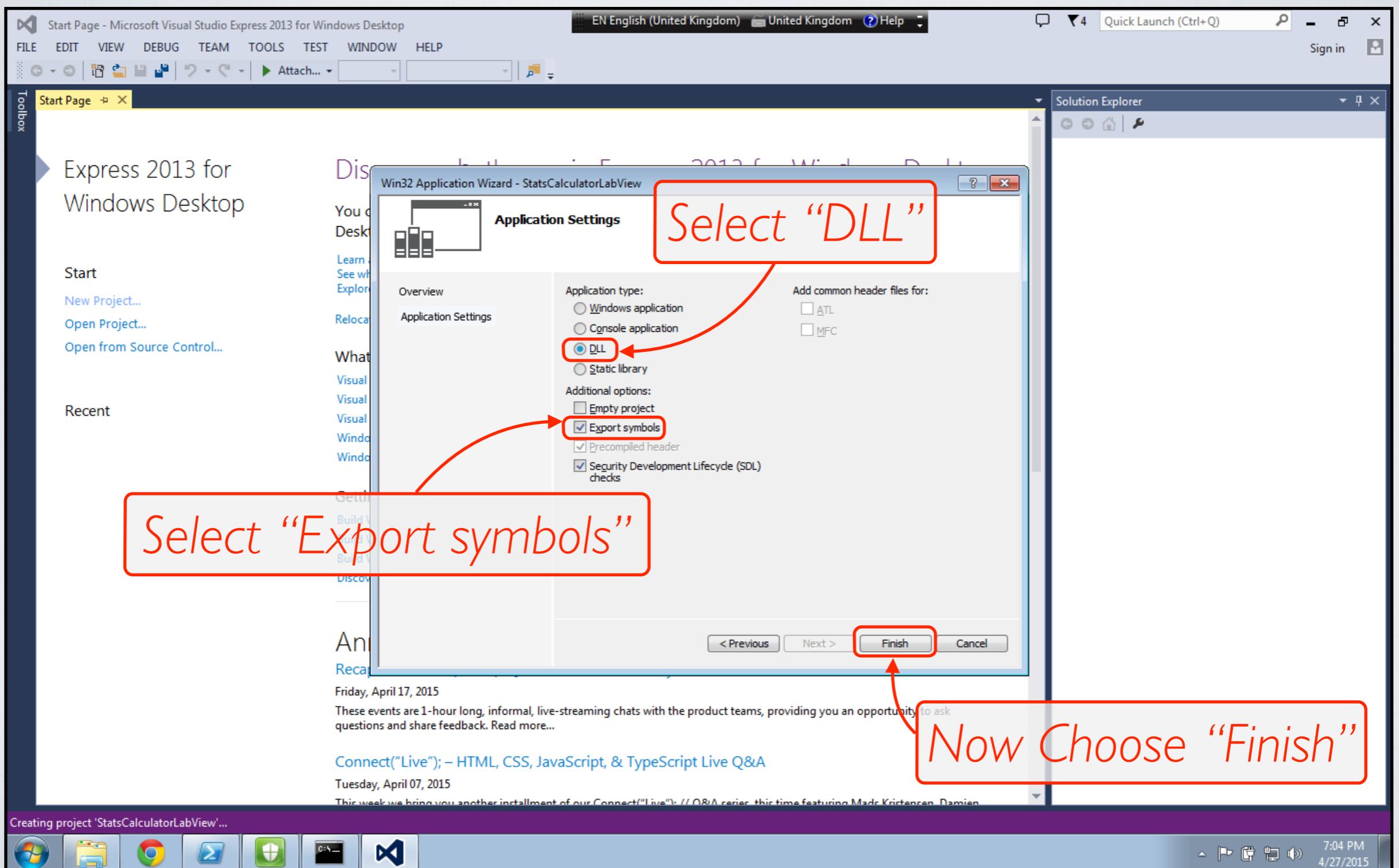
CREATE NEW PROJECT



CREATE NEW PROJECT



CREATE NEW PROJECT



STEP 3: MODIFY BOILERPLATE CODE

MODIFY BOILERPLATE CODE

(*StatsCalculatorLabView.cpp*)

StatsCalculatorLabView - Microsoft Visual Studio Express 2013 for Windows Desktop EN English (United Kingdom) United Kingdom Help

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP Local Windows Debugger Debug Win32

Solution Explorer

Search Solution Explorer (Ctrl+Shift+F)

Solution 'StatsCalculatorLabView' (1 project)

StatsCalculatorLabView

- External Dependencies
- Header Files
 - StatsCalculatorLabView.h
 - stdafx.h
 - targetver.h
- Resource Files
- Source Files
 - dllmain.cpp
 - StatsCalculatorLabView.cpp
 - stdafx.cpp
- ReadMe.txt

Properties

StatsCalculatorLabView Project Properties

Misc

| | |
|----------------------|---------------------------------|
| (Name) | StatsCalculatorLabView |
| Project Dependencies | |
| Project File | c:\Users\computationalphysics\c |
| Root Namespace | StatsCalculatorLabView |

(Name)
Specifies the project name.

Comment out all code

This item does not support previewing

Ln1 Col1 Ch1 INS

5:31 PM 4/28/2015

MODIFY BOILERPLATE CODE

(*StatsCalculatorLabView.h*)

The screenshot shows the Microsoft Visual Studio Express 2013 interface with the file *StatsCalculatorLabView.h* open. The code editor displays boilerplate code for a DLL. A red box highlights the preprocessor directive `#ifdef __cplusplus`. A red callout box labeled "Insert Preprocessor Directive" points to this line. Another red callout box labeled "Comment out code" points to the class definition block, which is enclosed in a dashed red rectangle.

```
#ifdef __cplusplus
#define STATSCALULATORLABVIEW_CAPI extern "C" __declspec(dllexport)
#endif

#ifndef STATSCALULATORLABVIEW_EXPORTS
#define STATSCALULATORLABVIEW_API __declspec(dllexport)
#endif

#ifdef __cplusplus
#define STATSCALULATORLABVIEW_CAPI extern "C" __declspec(dllexport)
#endif

#else
#define STATSCALULATORLABVIEW_API __declspec(dllimport)
#endif

/*
// This class is exported from the StatsCalculatorLabView.dll
class STATSCALULATORLABVIEW_API CStatsCalculatorLabView {
public:
    CStatsCalculatorLabView(void);
    // TODO: add your methods here.
};

extern STATSCALULATORLABVIEW_API int nStatsCalculatorLabView;

STATSCALULATORLABVIEW_API int fnStatsCalculatorLabView(void);
*/
```

The Solution Explorer on the right shows the project structure:

- Header Files
 - StatsCalculator.h
 - StatsCalculatorLabView.h
 - stdafx.h
 - targetver.h
- Resource Files
- Source Files
 - dllmain.cpp
 - StatsCalculator.cpp
 - StatsCalculatorCAPI.cpp
 - StatsCalculatorLabView.cpp
 - stdafx.cpp
- ReadMe.txt

The Properties window for *StatsCalculatorLabView.h* shows the following settings:

| | |
|---------------------|--------------------------------|
| Misc | |
| (Name) | StatsCalculatorLabView.h |
| Content | False |
| File Type | C++ Header File |
| Full Path | c:\Users\computationalphysics\ |
| Included In Project | True |
| Relative Path | StatsCalculatorLabView.h |

Annotations in red boxes:

- "Insert Preprocessor Directive" points to the first `#ifdef __cplusplus` line.
- "Comment out code" points to the class definition block, which is enclosed in a dashed red rectangle.

MODIFY BOILERPLATE CODE

(*stdafx.h*)

StdCalculatorLabView - Microsoft Visual Studio Express 2013 for Windows Desktop EN English (United Kingdom) United Kingdom Help

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP

Local Windows Debugger Debug Win32

Toolbox

stdafx.h* (Global Scope)

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//
#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
#include <windows.h>

// TODO: reference additional headers your program requires here
#include "StatsCalculator.h"
```

Insert Include Statement

Search Solution Explorer (Ctrl+Shift+F)

StatsCalculatorLabView

- External Dependencies
- Header Files
 - StatsCalculator.h
 - StatsCalculatorLabView.h
 - stdafx.h
 - targetver.h
- Resource Files
- Source Files
 - dllmain.cpp
 - StatsCalculator.cpp
 - StatsCalculatorCAPI.cpp
 - StatsCalculatorLabView.cpp
 - stdafx.cpp
- ReadMe.txt

Properties

100 %

Ready Ln 17 Col 29 Ch 29 INS

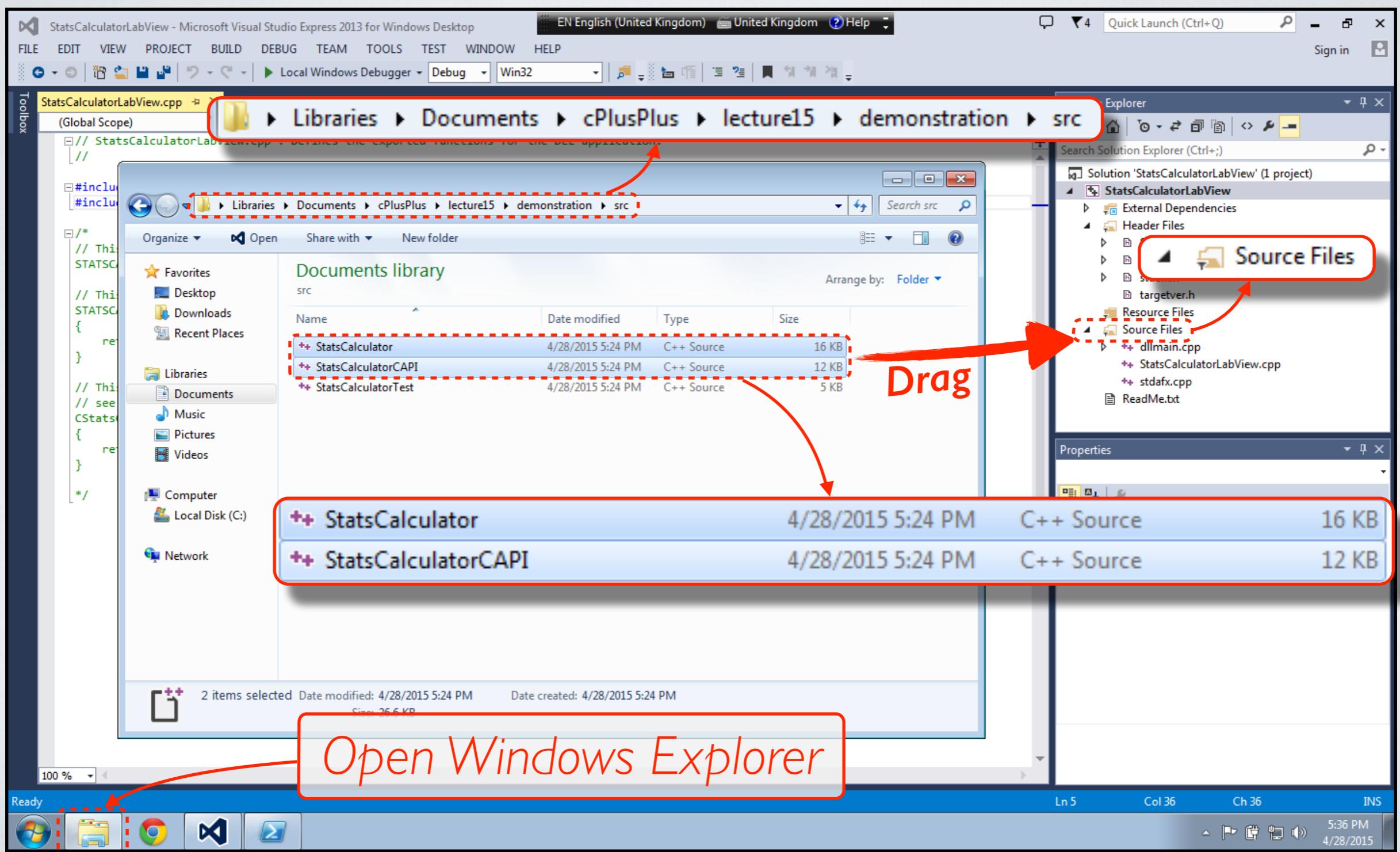
5:38 PM 4/28/2015

Error is expected. *StatsCalculator.h* must be added to the project.

STEP 4:
ADD NEW CODE

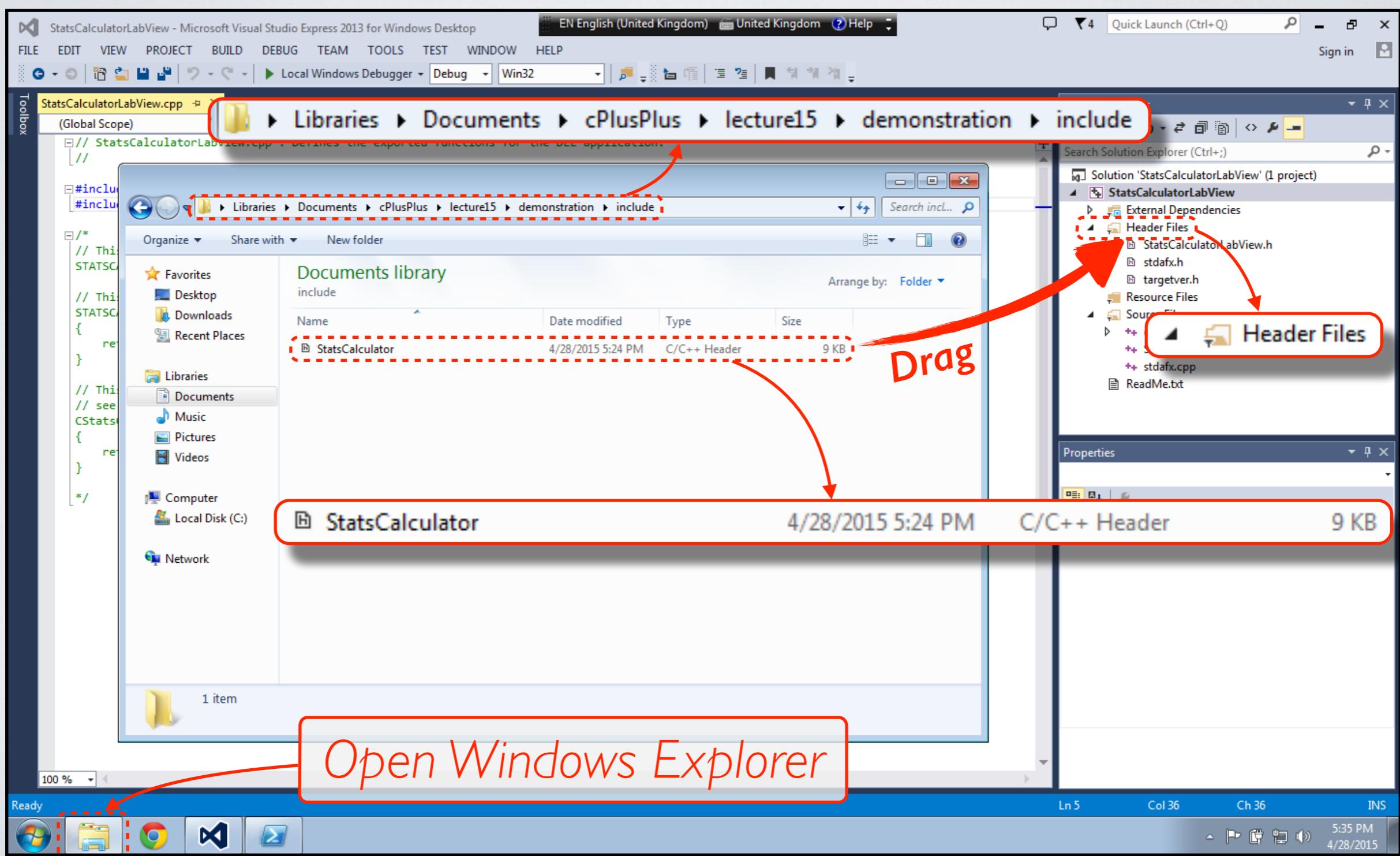
ADD NEW CODE

(*StatsCalculator.cpp, StatsCalculatorCAPI.cpp*)



ADD NEW CODE

(StatsCalculator.h)



STEP 4: MODIFY NEW CODE

MODIFY NEW CODE

(*StatsCalculator.h*)

The screenshot shows the Microsoft Visual Studio Express 2013 interface with the project 'StatsCalculatorLabView' open. The code editor displays the file *StatsCalculator.h*. A red callout box highlights the line `// Include StatsCalculatorLabView.h to provide preprocessor macros` and the include statement `#include "StatsCalculatorLabView.h"`. Another red callout box highlights the `#include <vector>` statement. The Solution Explorer pane shows the project structure, and the Properties pane displays file properties for *StatsCalculator.h*.

```
// Define the STATSCALULATOR_H macro to act as an include guard
#ifndef STATSCALULATOR_H
#define STATSCALULATOR_H

// Include StatsCalculatorLabView.h to provide preprocessor macros
#include "StatsCalculatorLabView.h"

/* This header file may be parsed by a C compiler. If this happens, any
 * C++-only language constructs (e.g. class definitions or references to
 * std::string or std::vector) in the code will cause the compilation to
 * fail.
 *
 * To prevent a C compiler from attempting to parse C++ code, wrap that code
 * in a PREPROCESSOR CONDITIONAL block that depends upon the "__cplusplus"
 * macro being defined.
 *
 * This will only be the case if a C++ compiler is considering the code.
 */
#ifndef __cplusplus

// Include the <vector> header to provide the STL std::vector type.
#include <vector>

// Insert Include Statement
// Include StatsCalculatorLabView.h to provide preprocessor macros
#include "StatsCalculatorLabView.h"

/*
 * methods to compute and return several statistical properties of those
 * numbers.
 *
 * It provides a method that prints a summary of the statistical properties
 * of its internally stored numbers to the terminal and another method that
 * writes a similar summary to a textual output file.
 */
class StatsCalculator {

    /** \brief An STL vector of double precision values to store parsed numeric
     * values.
     */

100 %
```

Insert Include Statement

Insert Include Statement

Properties

| | |
|-----------------------------------|--|
| StatsCalculator.h File Properties | |
| Misc | |
| (Name) | StatsCalculator.h |
| Content | False |
| File Type | C++ Header File |
| Full Path | c:\Users\computationalphysics\cPlusPlus\lecture15\demo\StatsCalculator.h |
| Included In Project | True |
| Relative Path | ..\..\..\cPlusPlus\lecture15\demo\StatsCalculator.h |

(Name)
Names the file object.

MODIFY NEW CODE

(*StatsCalculator.h*)

STATSCALULATORLABVIEW_API

Prepend **all** declarations with **preprocessor macro.**

```
* "C".
*
* The final complication is that C compilers cannot parse the 'extern "C"' token
* pair, so a PREPROCESSOR CONDITIONAL that depends on the __cplusplus macro
* is required. Another conditional block is also required to hide the closing
* brace of the code block from C compilers.
*/
#ifndef __cplusplus
#define __cplusplus
#endif // __cplusplus was defined

/** \brief C API function that instantiates a StatsCalculator object and returns an
 * integer "handle" to the user that can be used to reference the object.
 *
 * \return An integer "handle" that uniquely refers to the instantiated
 * StatsCalculator object.
 */
STATSCALULATORLABVIEW_API int statsCalcCreate();

/** \brief Destroy a previously instantiated StatsCalculator object.
 *
 * \param handle - an integer that was returned by statsCalcCreate() and
 * uniquely references the instance of StatsCalculator that should
 * be destroyed.
 */
STATSCALULATORLABVIEW_API void statsCalcDestroy(int handle);

/** \brief Expose the functionality of StatsCalculator::appendValue() in the
 * C API.
 *
 * \param handle - an integer that was returned by statsCalcCreate() and
 * uniquely references the instance of StatsCalculator that should
 * be induced to parse the file.
 * \param value - A double precision value to append to the "numericValues"
 * member datum of the StatsCalculator instance to which handle refers.
 */
STATSCALULATORLABVIEW_API void statsCalcAppendValue(int handle, double value);

/** \brief Expose the functionality of StatsCalculator::readFile() in the
 * C API.
 *
 */
STATSCALULATORLABVIEW_API void statsCalcReadFile(int handle, const char* filename);
```

MODIFY NEW CODE

(*StatsCalculatorCAPI.cpp*)

Replace **all** occurrences
of **extern "C"** with
preprocessor macro.

STATSCALULATORLABVIEW_CAPI

Replace

```
/* which returns an iterator corresponding to the std::pair that was inserted. The C API function
 * then dereferences the iterator and returns the first element of the std::pair.
 *
 * Since the keys of a std::map must be unique, the returned integer provides a unique handle to a
 * particular StatsCalculator instance.
 *
 * \return An integer "handle" that uniquely refers to the instantiated
 * StatsCalculator object.
 */
STATSCALULATORLABVIEW_CAPI int statsCalcCreate(){
    // Compute an appropriate key to associate with a new StatsCalculator instance.
    int key = statsCalculators.empty() ? 0 : statsCalculators.rbegin()->first + 1;
    /* Construct a new std::pair<int, StatsCalculator> and insert it into the global "statsCalculators"
     * std::map.
     *
     * If the insertion is successful, an immutable iterator corresponding to the inserted element is
     * returned. Dereferencing this iterator returns an immutable reference to the ELEMENT, which
     * corresponds to a key-value pair.
     */
    const std::pair<int, StatsCalculator> & created = *statsCalculators.emplace(
        /* return the first element of the inserted std::pair, which corresponds
         * to key.
         */
        key,
        created);
    return created.first;
}

/** Searches the global "statsCalculators" for an element corresponding to the integer
 * handle that is provided as the function argument. If the corresponding element is
 * found, it is destroyed. Otherwise this function is a no-op.
 *
 * \param handle - an integer that was returned by statsCalcCreate() and
 * uniquely references the instance of StatsCalculator that should
 * be destroyed.
 */
extern "C" void statsCalcDestroy(int handle){
    std::map<int, StatsCalculator>::iterator handlePos = statsCalculators.find(handle);
    if(handlePos != statsCalculators.end()){
        statsCalculators.erase(handlePos);
    }
}
```

STATSCALULATORLABVIEW_CAPI void statsCalcDestroy(int handle){

extern "C" void statsCalcDestroy(int handle){

MODIFY NEW CODE

(*StatsCalculatorCAPI.cpp*)

StatsCalculatorLabView - Microsoft Visual Studio Express 2013 for Windows Desktop EN English (United Kingdom) United Kingdom Help

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP

Local Windows Debugger Debug x64

Toolbox

Solution Explorer

Search Solution Explorer (Ctrl+Shift+F)

StatsCalculatorLabView

- External Dependencies
- Header Files
 - StatsCalculator.h
 - StatsCalculatorLabView.h
 - stdafx.h
 - targetver.h
- Resource Files
- Source Files
 - dllmain.cpp
 - StatsCalculator.cpp
 - StatsCalculatorCAPI.cpp
 - StatsCalculatorLabView.cpp
 - stdafx.cpp
- ReadMe.txt

Properties

Ready Ln1 Col1 Ch1 INS

6:36 PM 4/28/2015

```
/* StatsCalculatorCAPI.cpp Definition of the C API for the StatsCalculator class.
```

MODIFY NEW CODE

(*StatsCalculator.cpp*)

The screenshot shows the Microsoft Visual Studio Express 2013 interface. The main window displays the file *StatsCalculator.cpp*. A red callout box with the text "Insert Include Statement" points to the line `#include "stdafx.h"`. Another red callout box highlights the same line in the code editor. The Solution Explorer on the right shows the project structure, including files like *StatsCalculator.h*, *stdafx.h*, and *targetver.h*.

```
// IMPLEMENTATION file for StatsCalculator class

// STL HEADER FILES
// The <cmath> header is included to provide the std::sqrt(...) function.
#include <cmath>
// The <fstream> header is included to enable input from and output to files.
#include <fstream>
// The <iostream> header is included to enable textual terminal output.
#include <iostream>

// LOCAL HEADER FILES
// Include the stdafx.h header to satisfy Windows requirements
#include "stdafx.h"

/* The "StatsCalculator.h" header is included to provide a definition of the
 * StatsCalculator class.
 */
#include "StatsCalculator.h"

// PUBLIC METHODS OF STATSCALULATOR

/*
 * // Include the stdafx.h header to satisfy Windows requirements
 * #include "stdafx.h"
 */

// If any numeric values were successfully parsed from the input file...
if(numericValues.size() > 0){

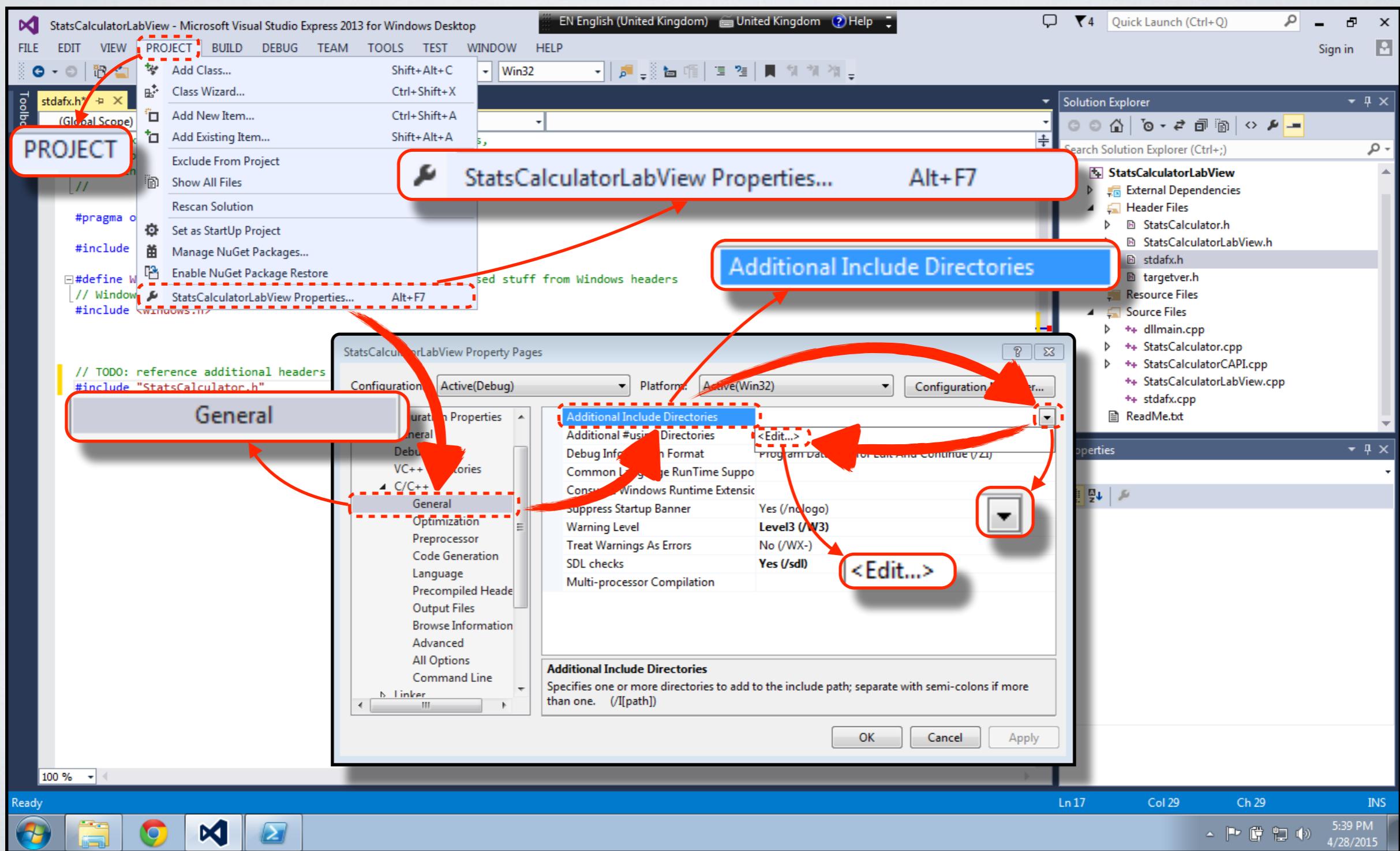
    /* Declare and zero-initialize a double precision variable with
     * identifier "sum" store the computed sum
     */
    double sum(0.0);

    /* Use the range-based for-loop syntax to iterate over all elements
     * of the "numericValues" member datum.
     */
}
```

STEP 4: MODIFY PROJECT SETTINGS

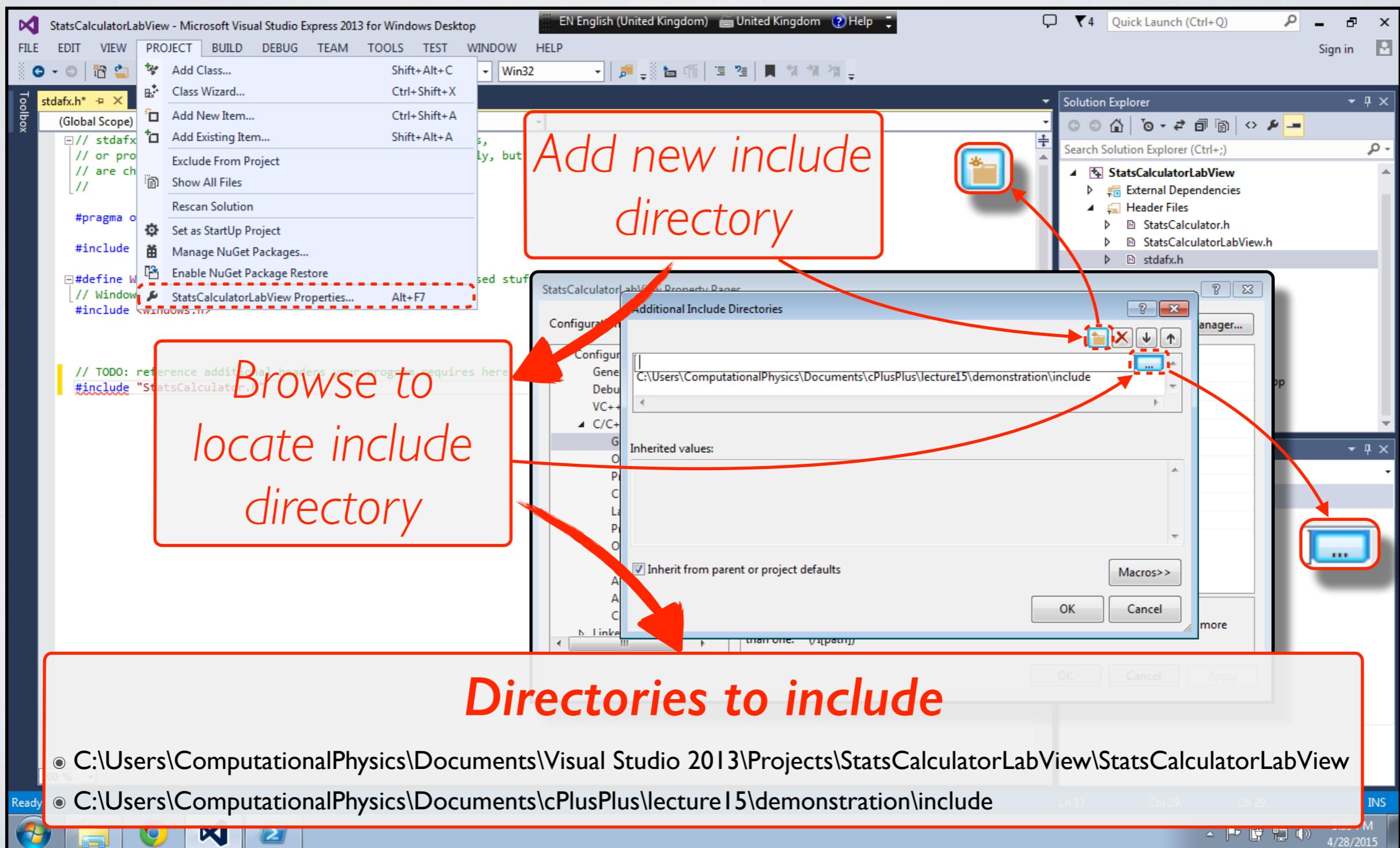
MODIFY PROJECT SETTINGS

ADD INCLUDE DIRECTORIES



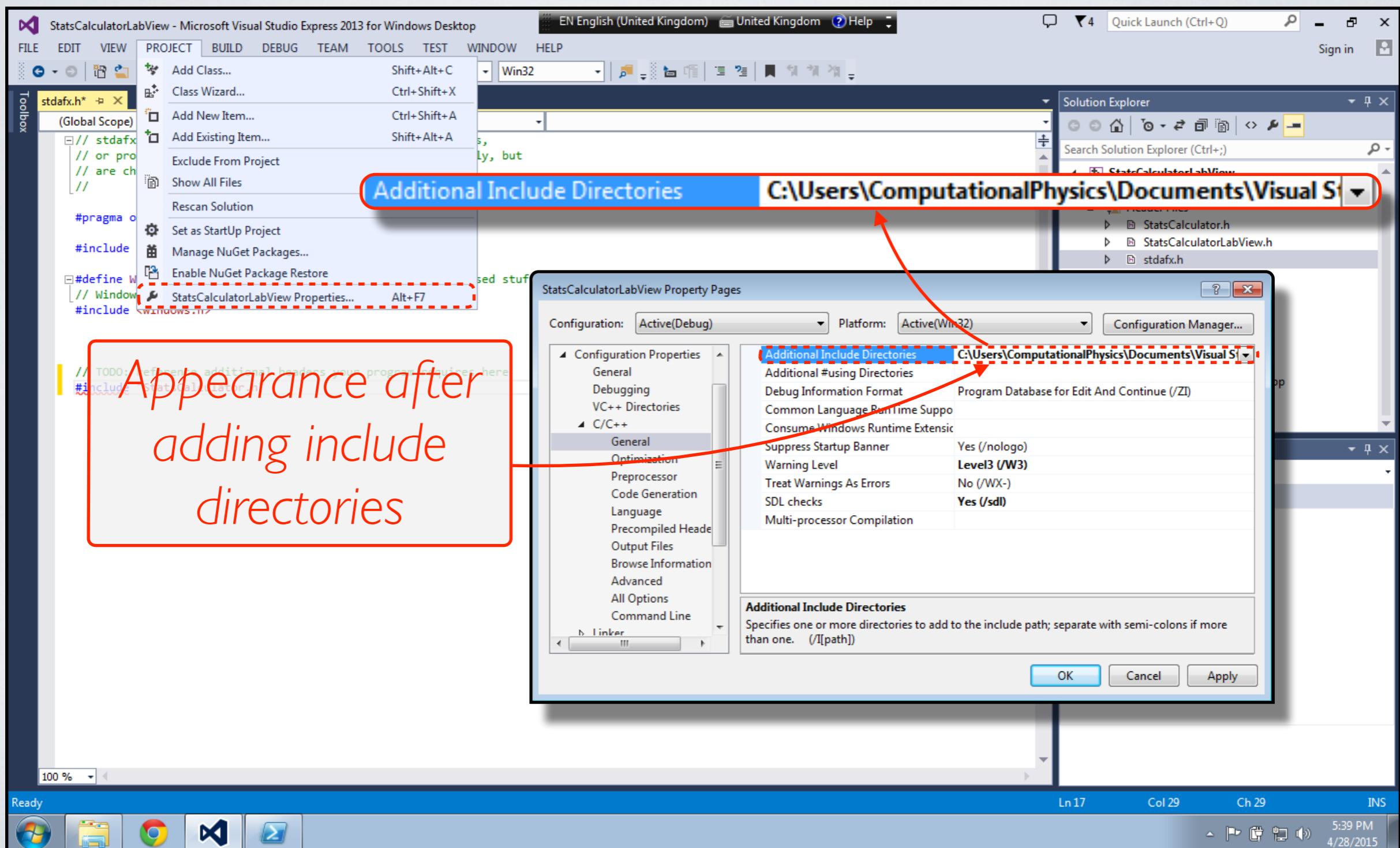
MODIFY PROJECT SETTINGS

ADD INCLUDE DIRECTORIES



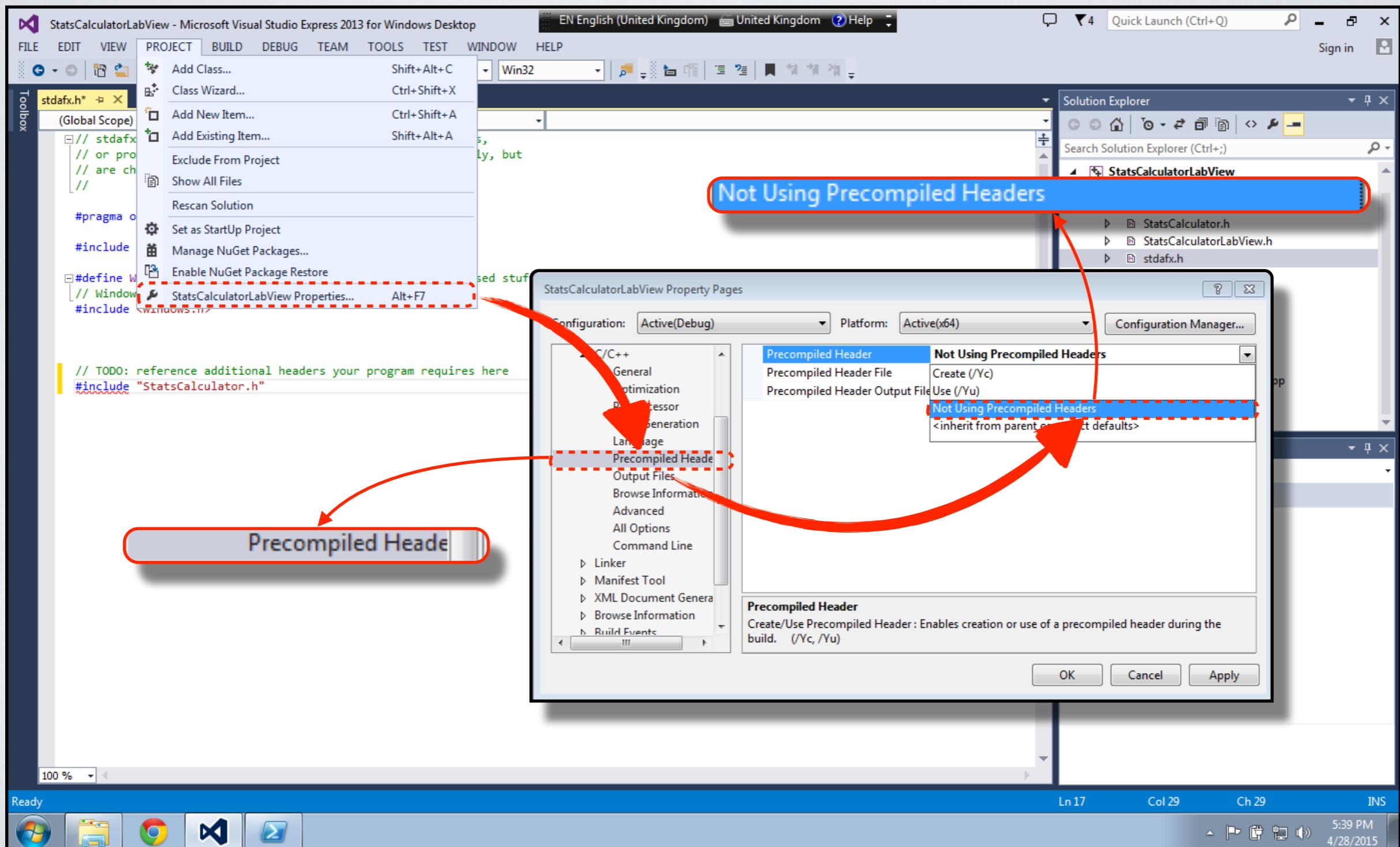
MODIFY PROJECT SETTINGS

ADD INCLUDE DIRECTORIES



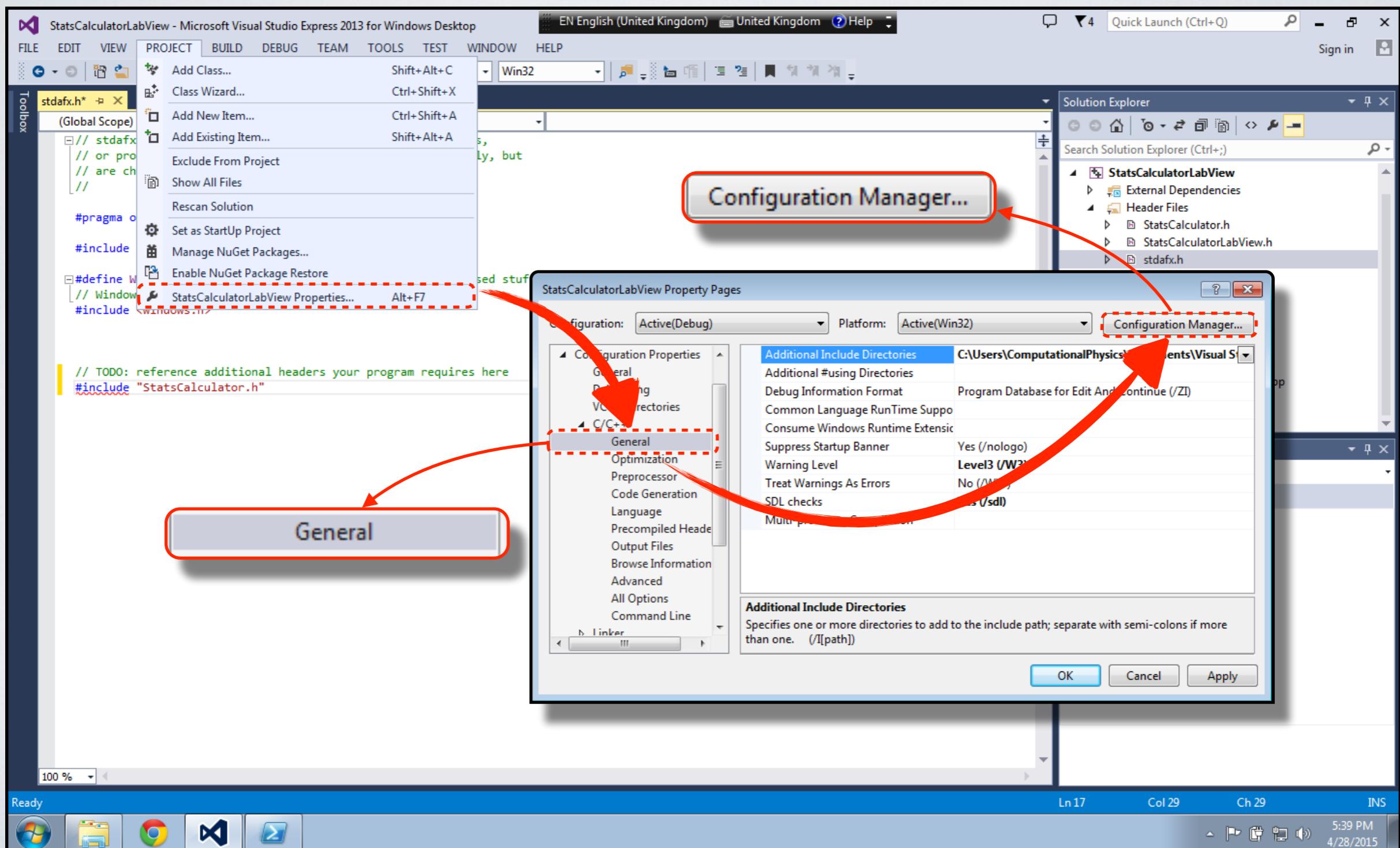
MODIFY PROJECT SETTINGS

DISABLE PRECOMPILED HEADER USAGE



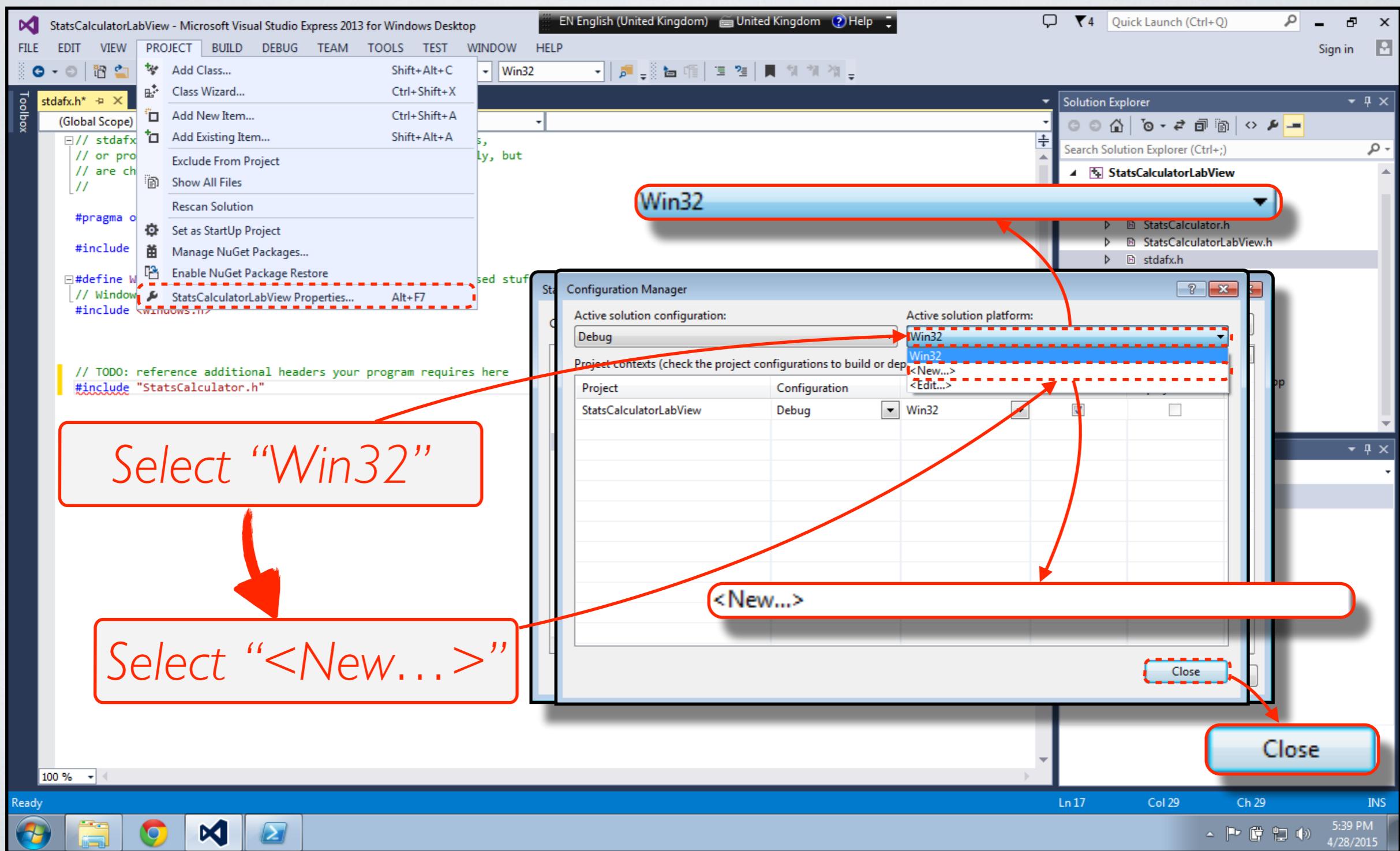
MODIFY PROJECT SETTINGS

ENABLE 64-BIT CODE GENERATION



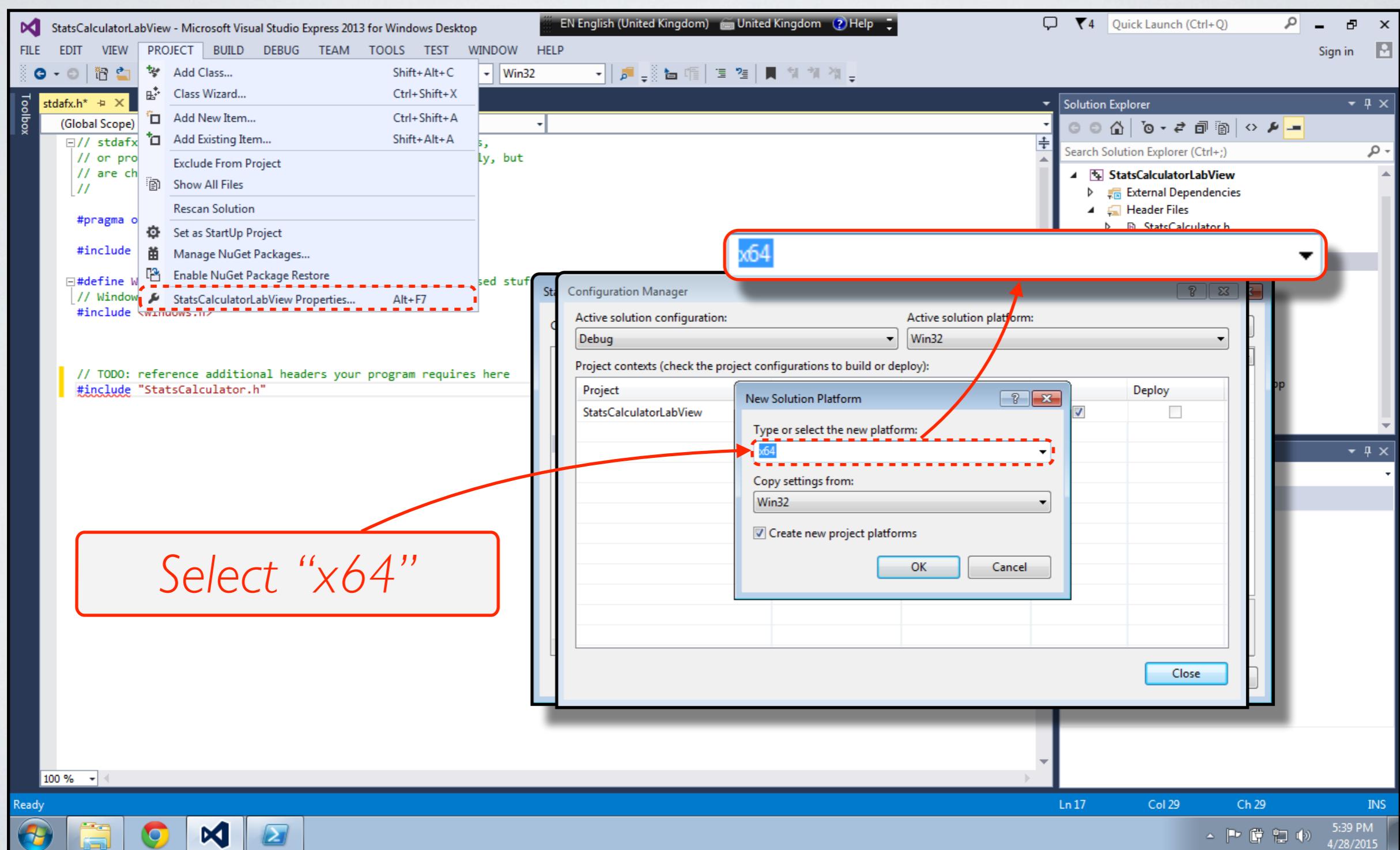
MODIFY PROJECT SETTINGS

ENABLE 64-BIT CODE GENERATION



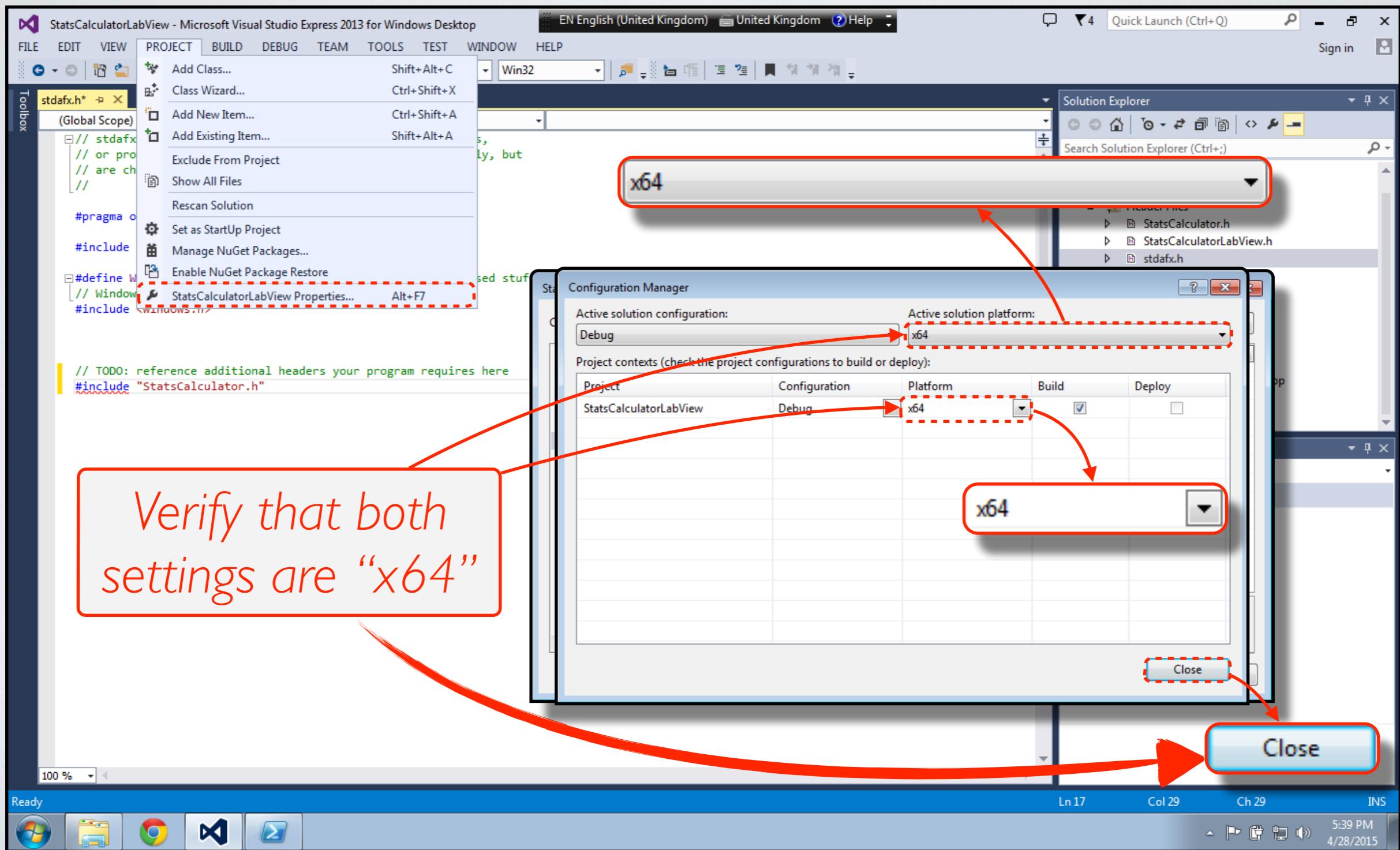
MODIFY PROJECT SETTINGS

ENABLE 64-BIT CODE GENERATION



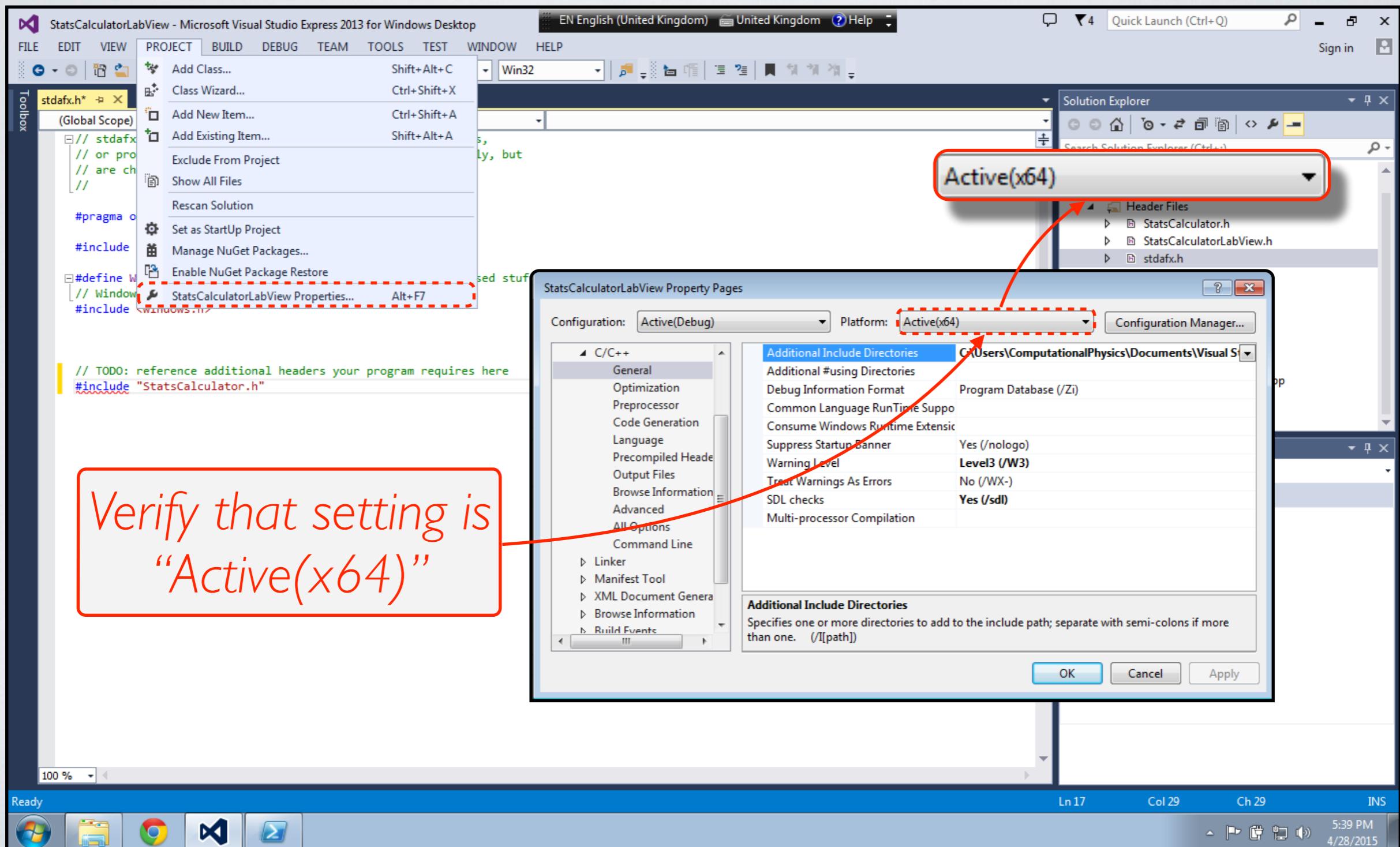
MODIFY PROJECT SETTINGS

ENABLE 64-BIT CODE GENERATION



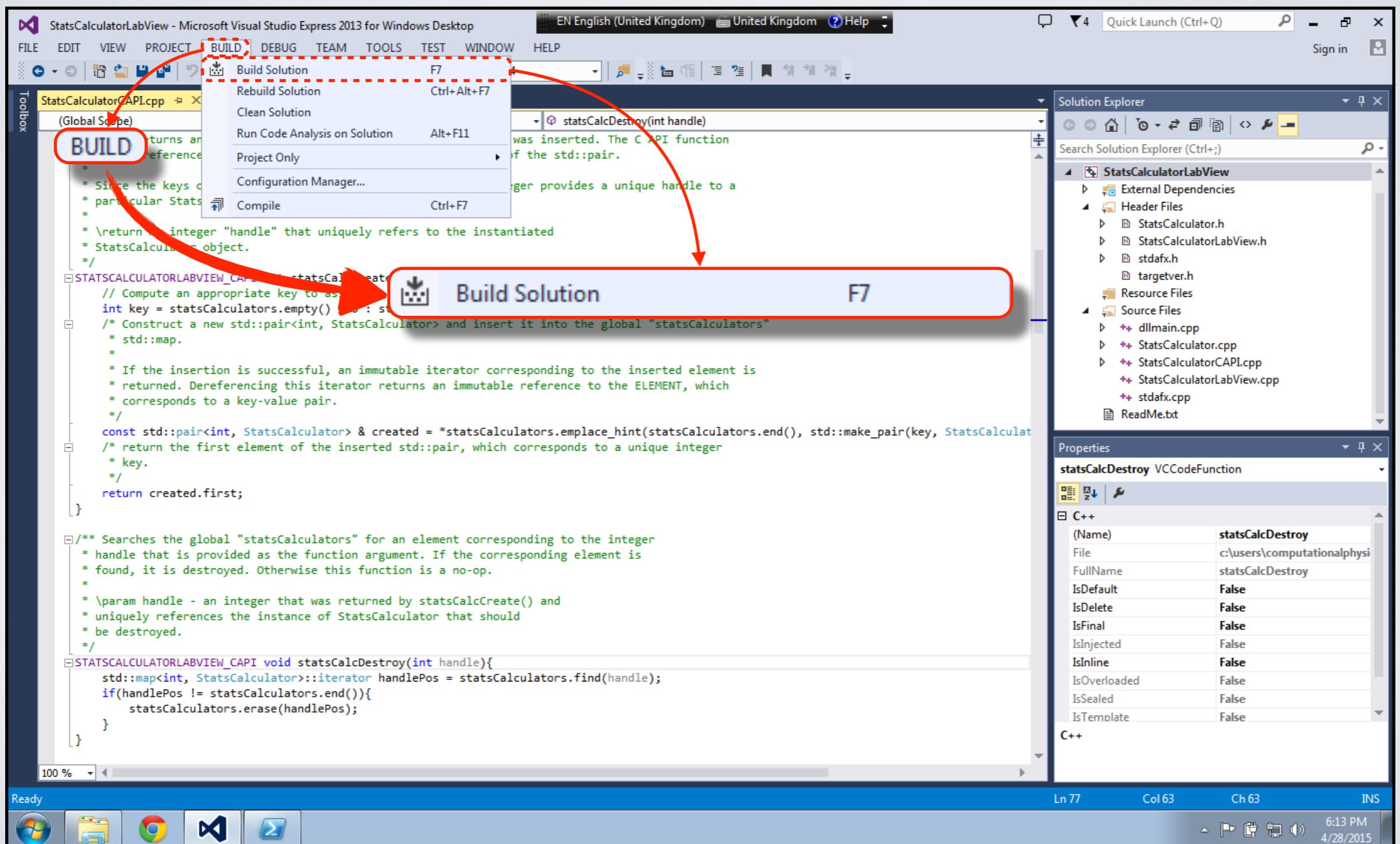
MODIFY PROJECT SETTINGS

ENABLE 64-BIT CODE GENERATION

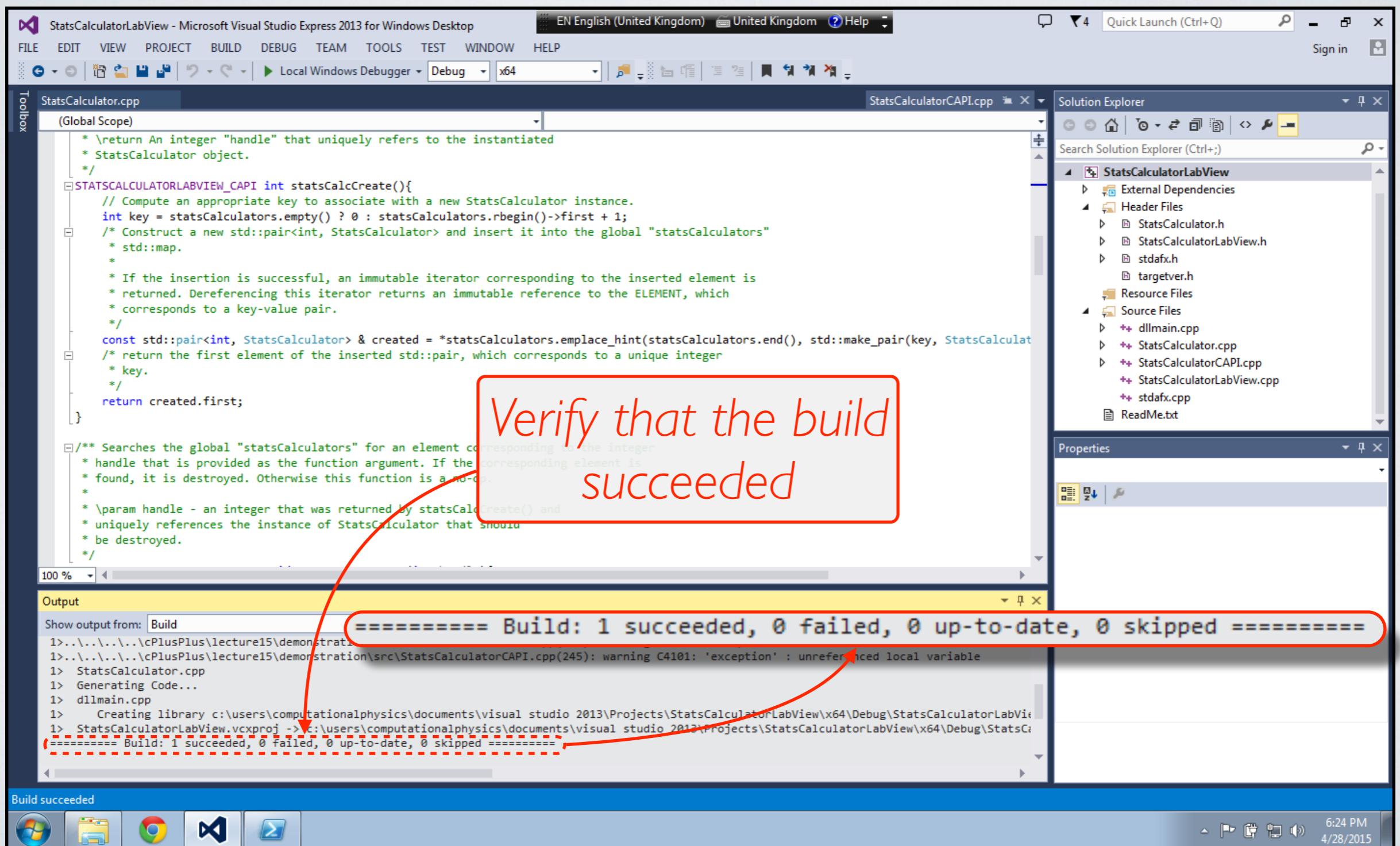


STEP 5: BUILD PROJECT

BUILD PROJECT

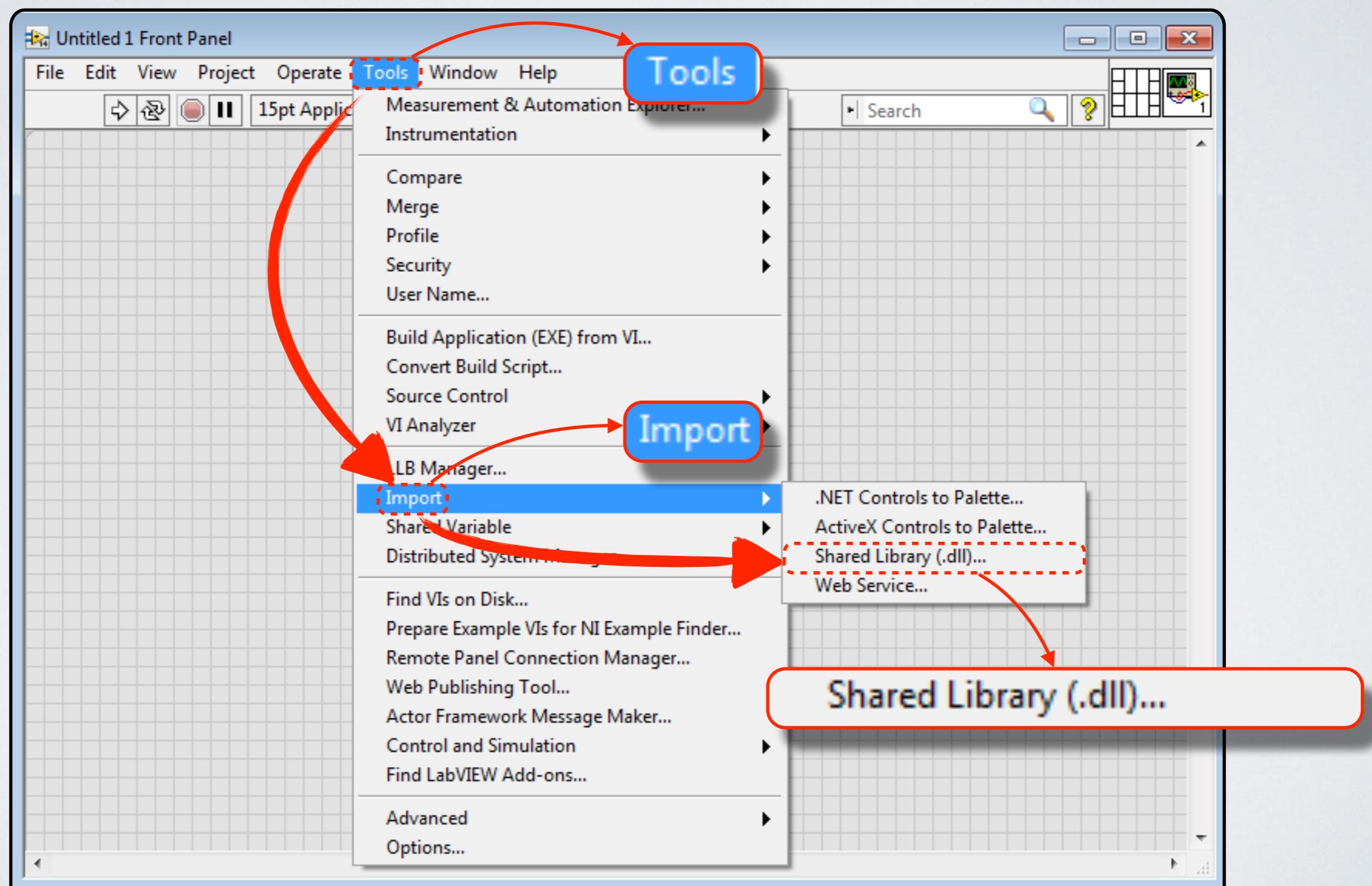


BUILD PROJECT



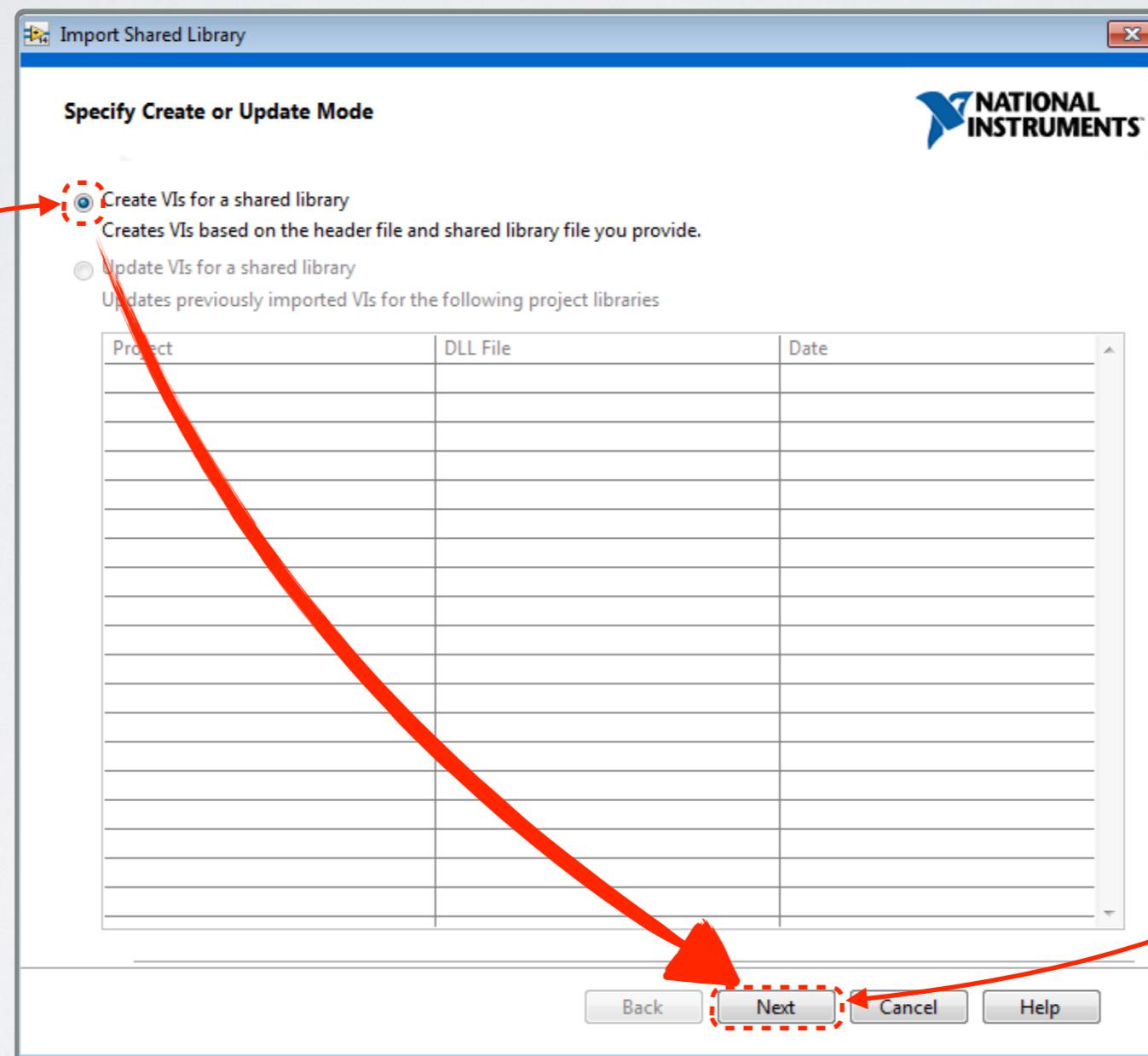
STEP 6: IMPORT THE LIBRARY

IMPORT SHARED LIBRARY USING THE IMPORT WIZARD TOOL



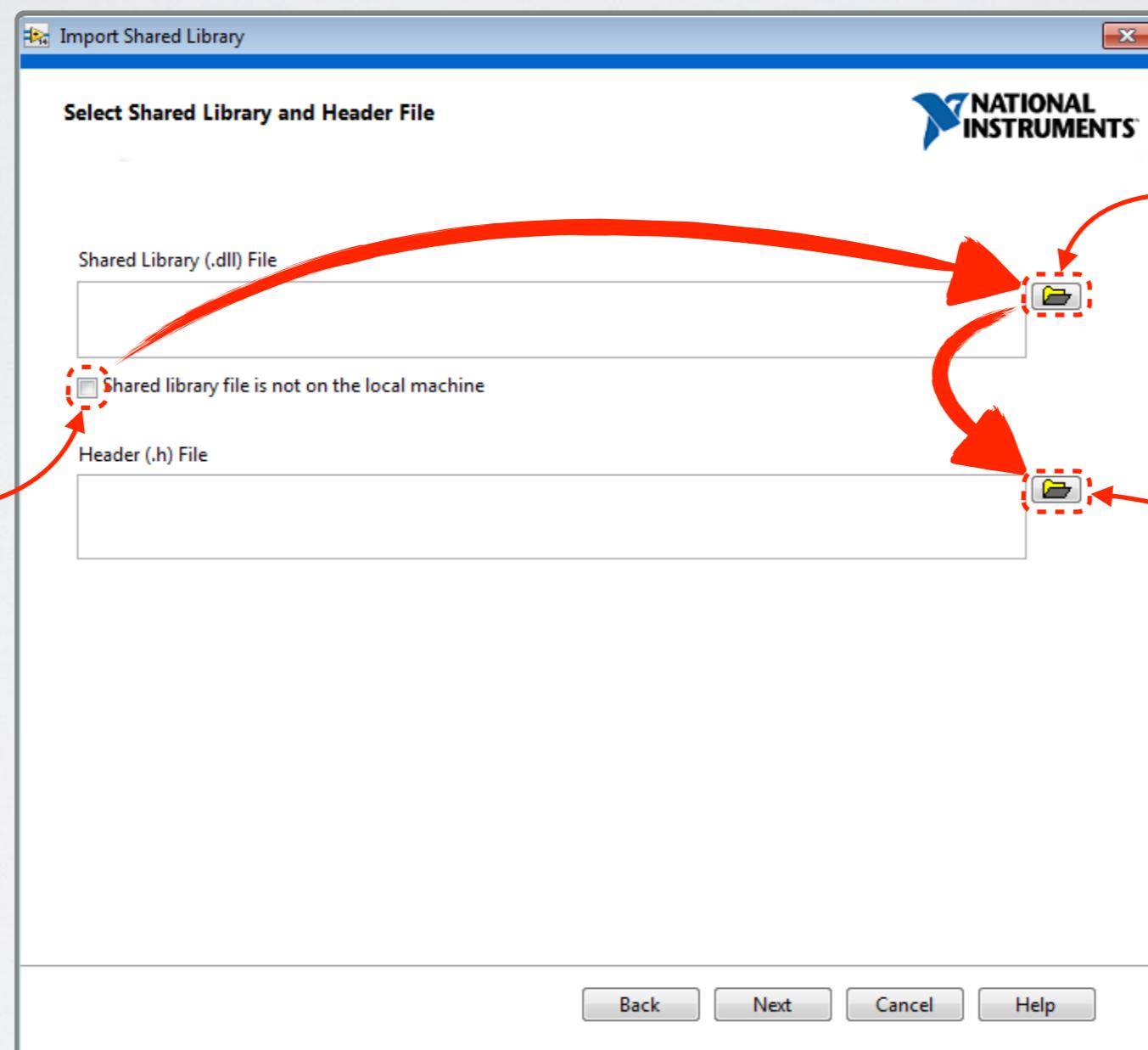
IMPORT SHARED LIBRARY

SPECIFY CREATE OR UPDATE MODE



IMPORT SHARED LIBRARY

SELECT SHARED LIBRARY AND HEADER FILE



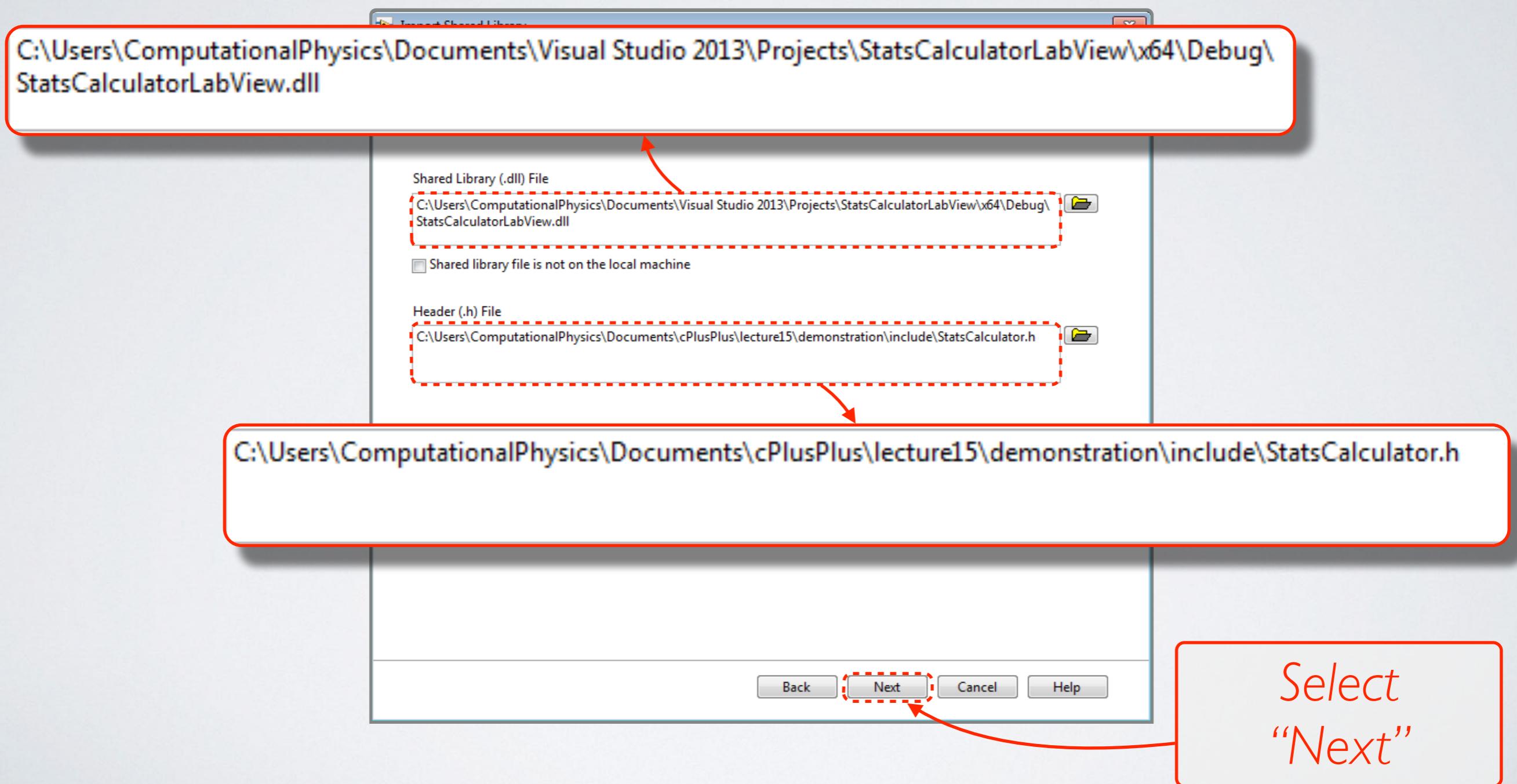
Verify that
checkbox is
not selected.

Browse for
shared library
file.

Browse for
header file
that defines
the exported
symbols.

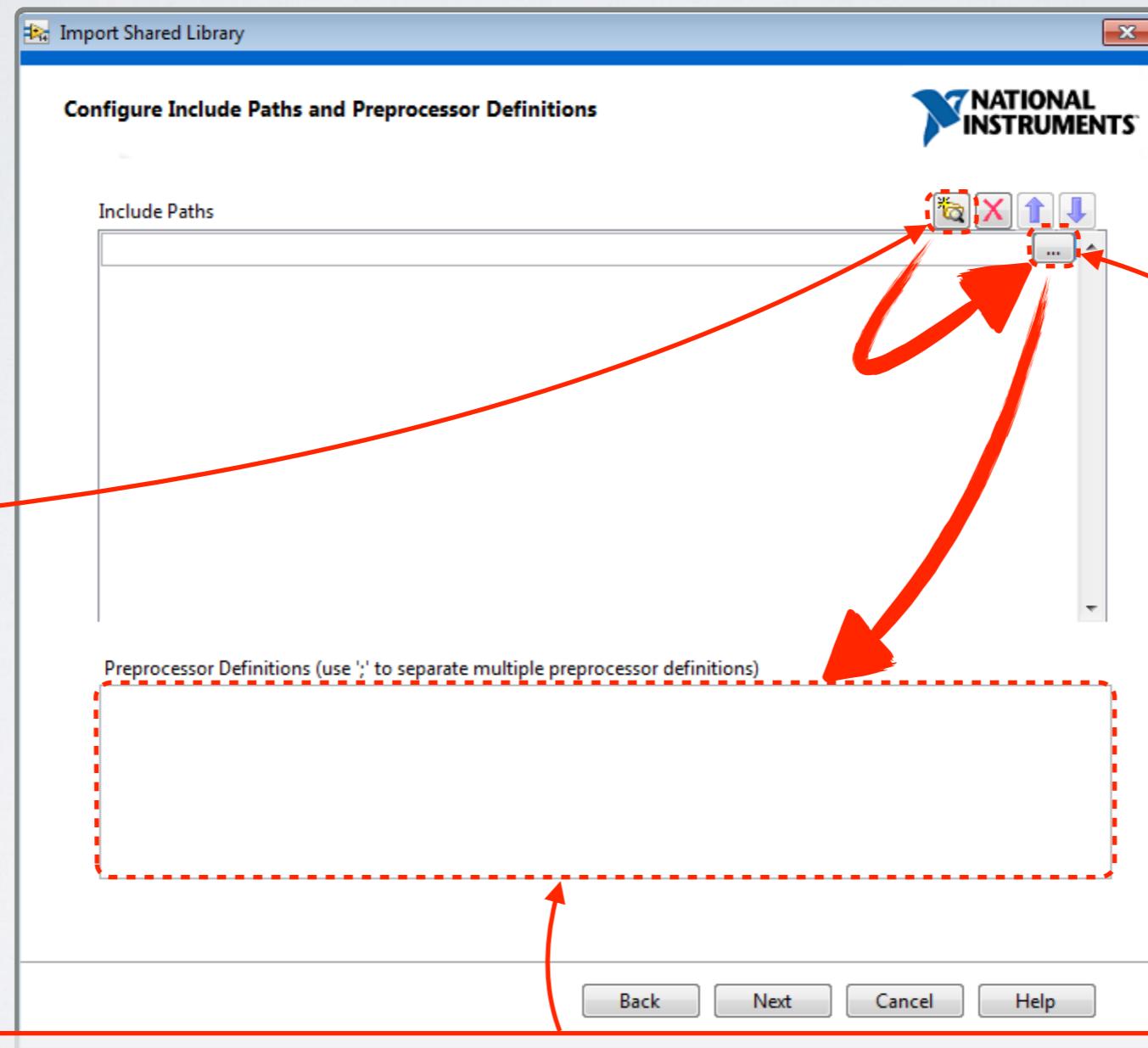
IMPORT SHARED LIBRARY

SELECT SHARED LIBRARY AND HEADER FILE



IMPORT SHARED LIBRARY

CONFIGURE INCLUDE PATHS AND PREPROCESSOR DEFINITIONS



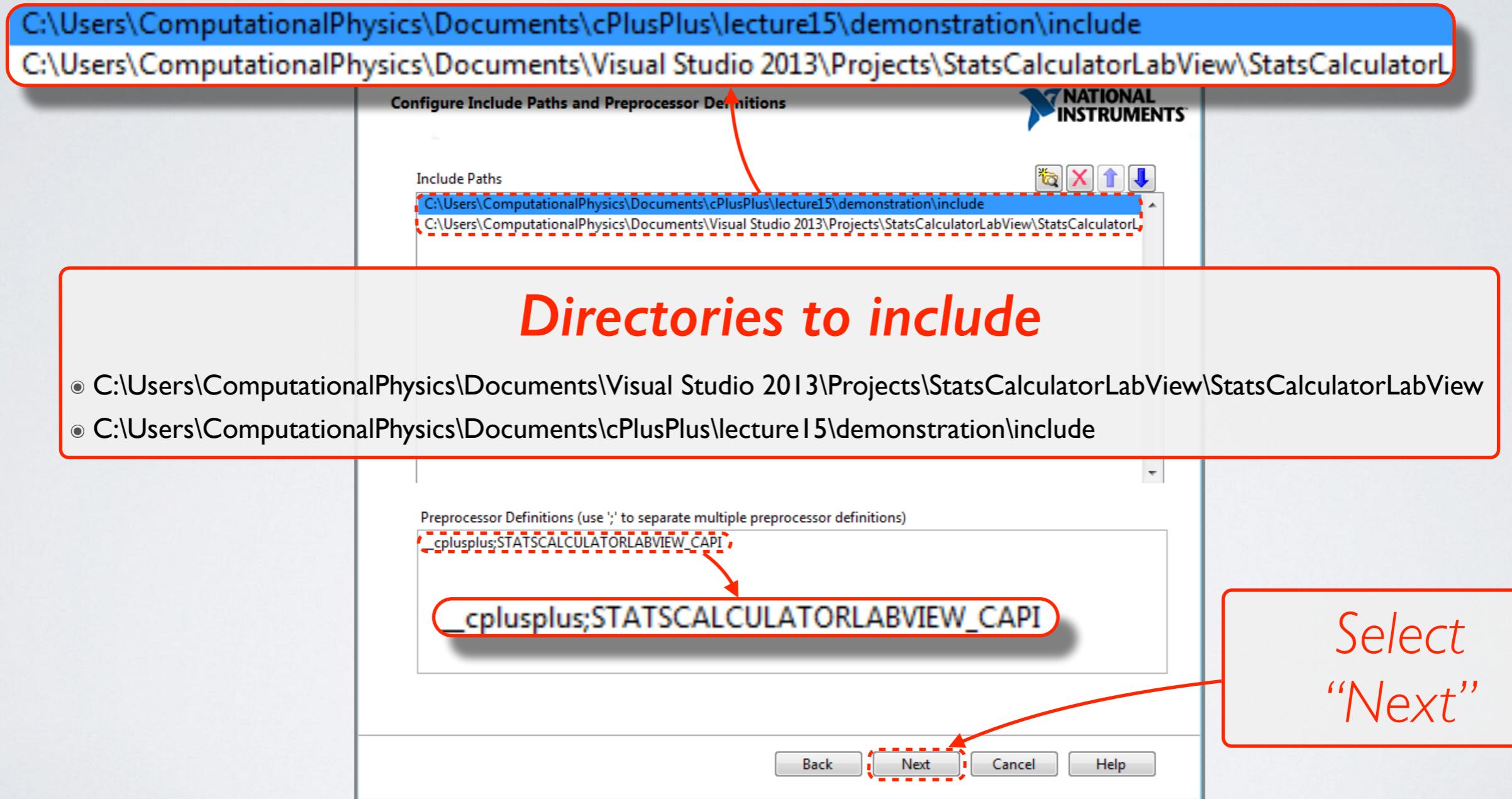
Add new
Include Path.

Browse for
Include Paths.

Add preprocessor definitions that make functions visible.

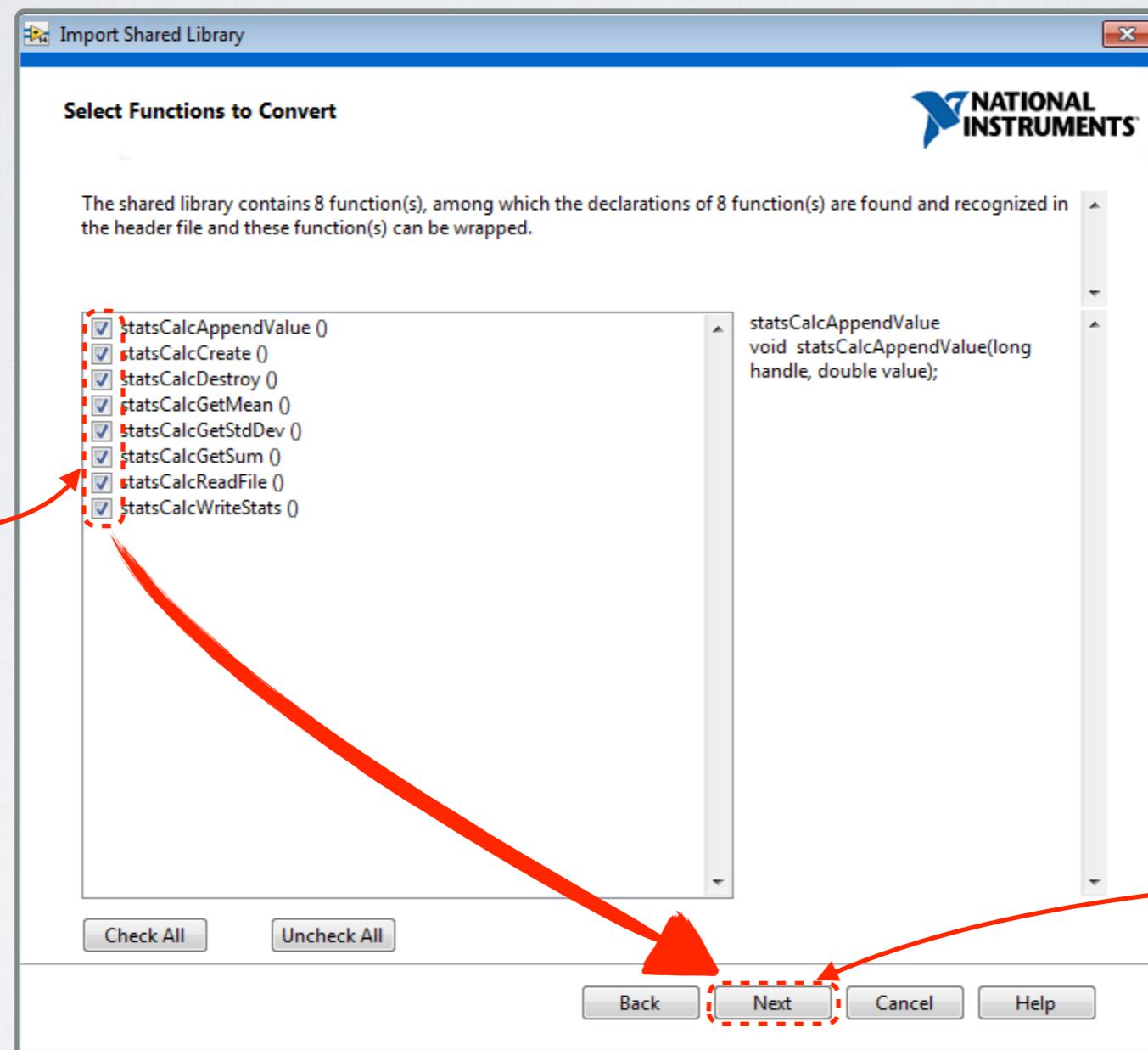
IMPORT SHARED LIBRARY

CONFIGURE INCLUDE PATHS AND PREPROCESSOR DEFINITIONS



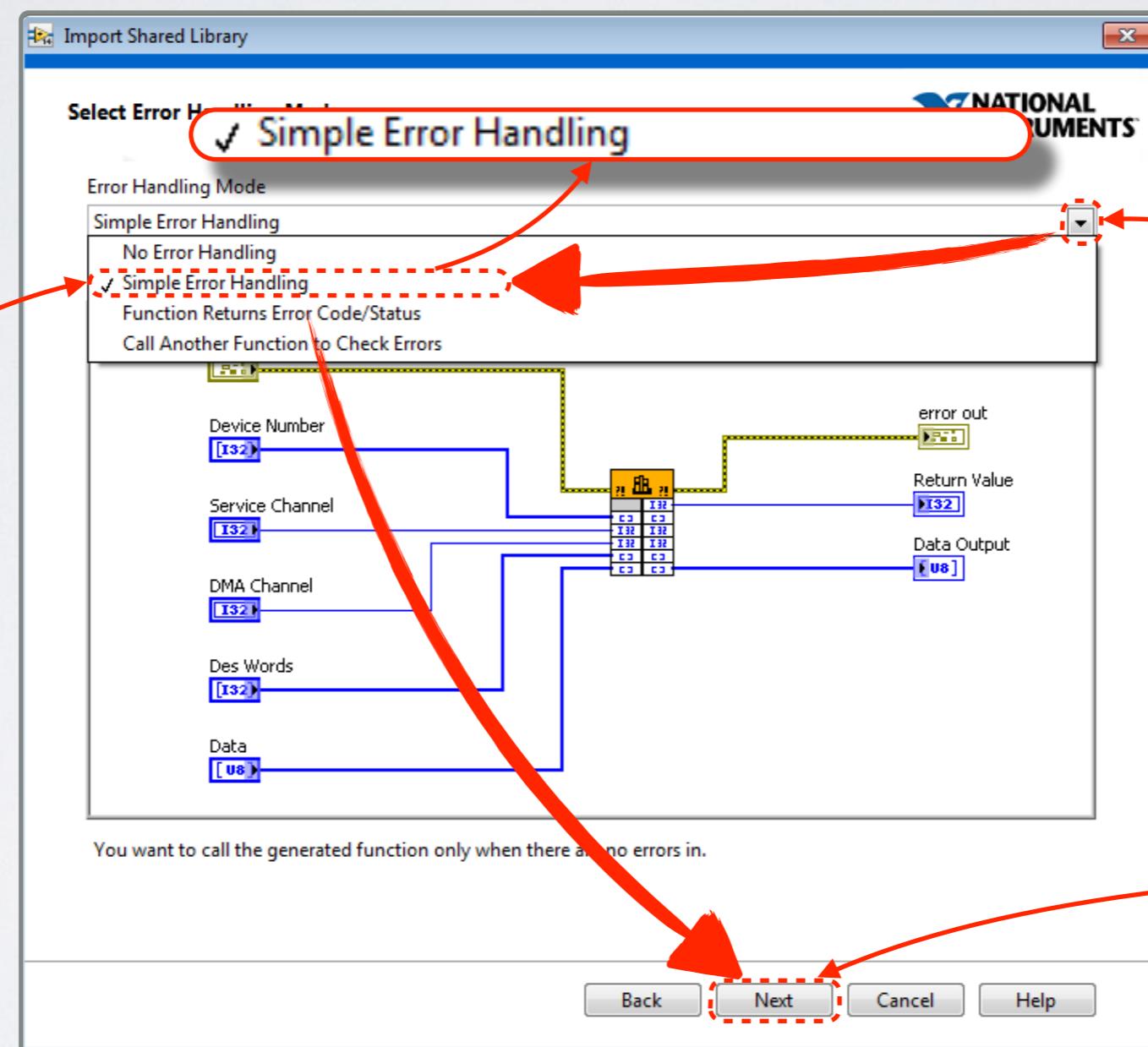
IMPORT SHARED LIBRARY

SELECT FUNCTIONS TO CONVERT



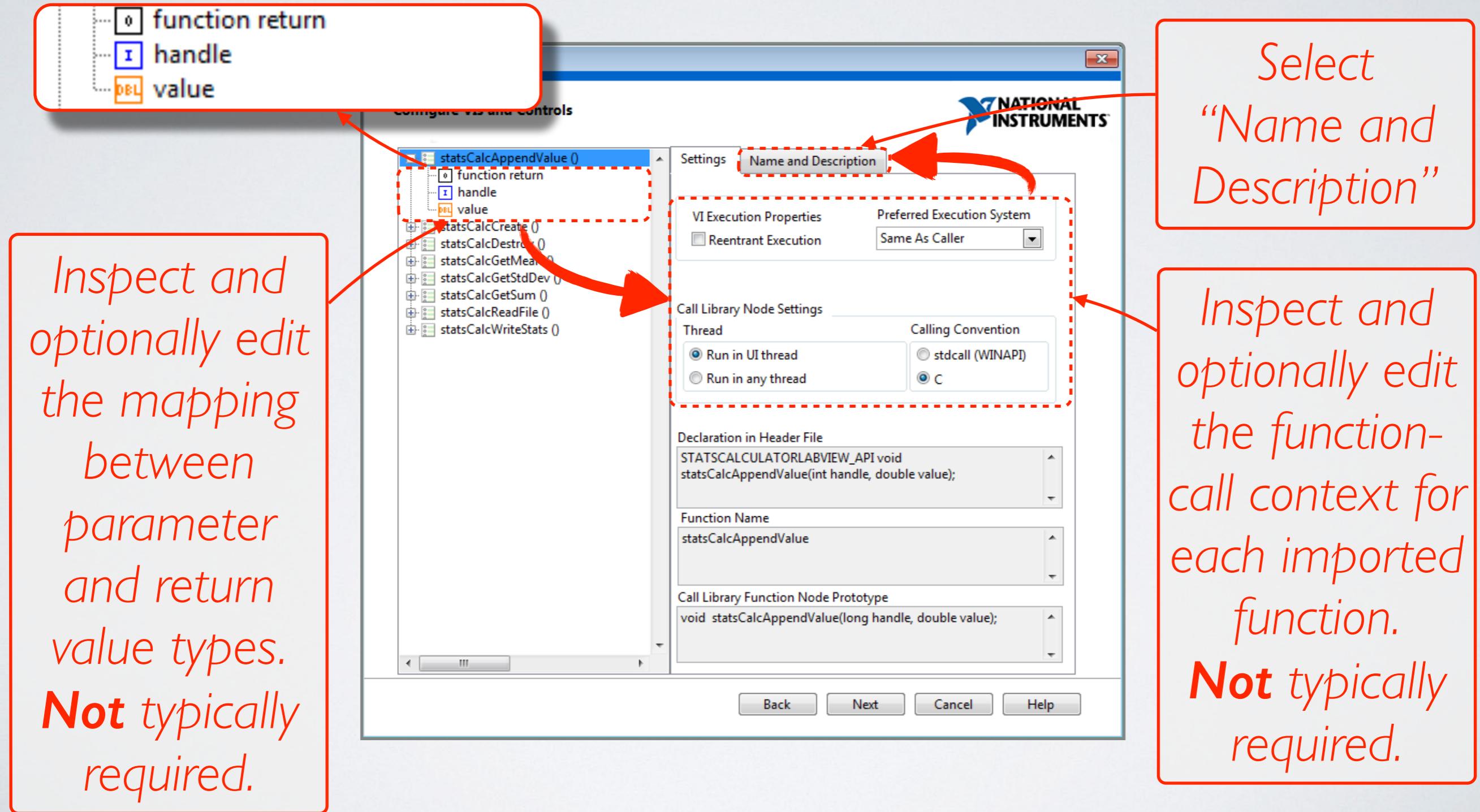
IMPORT SHARED LIBRARY

SELECT ERROR HANDLING MODE



IMPORT SHARED LIBRARY

CONFIGURE VIS AND CONTROLS

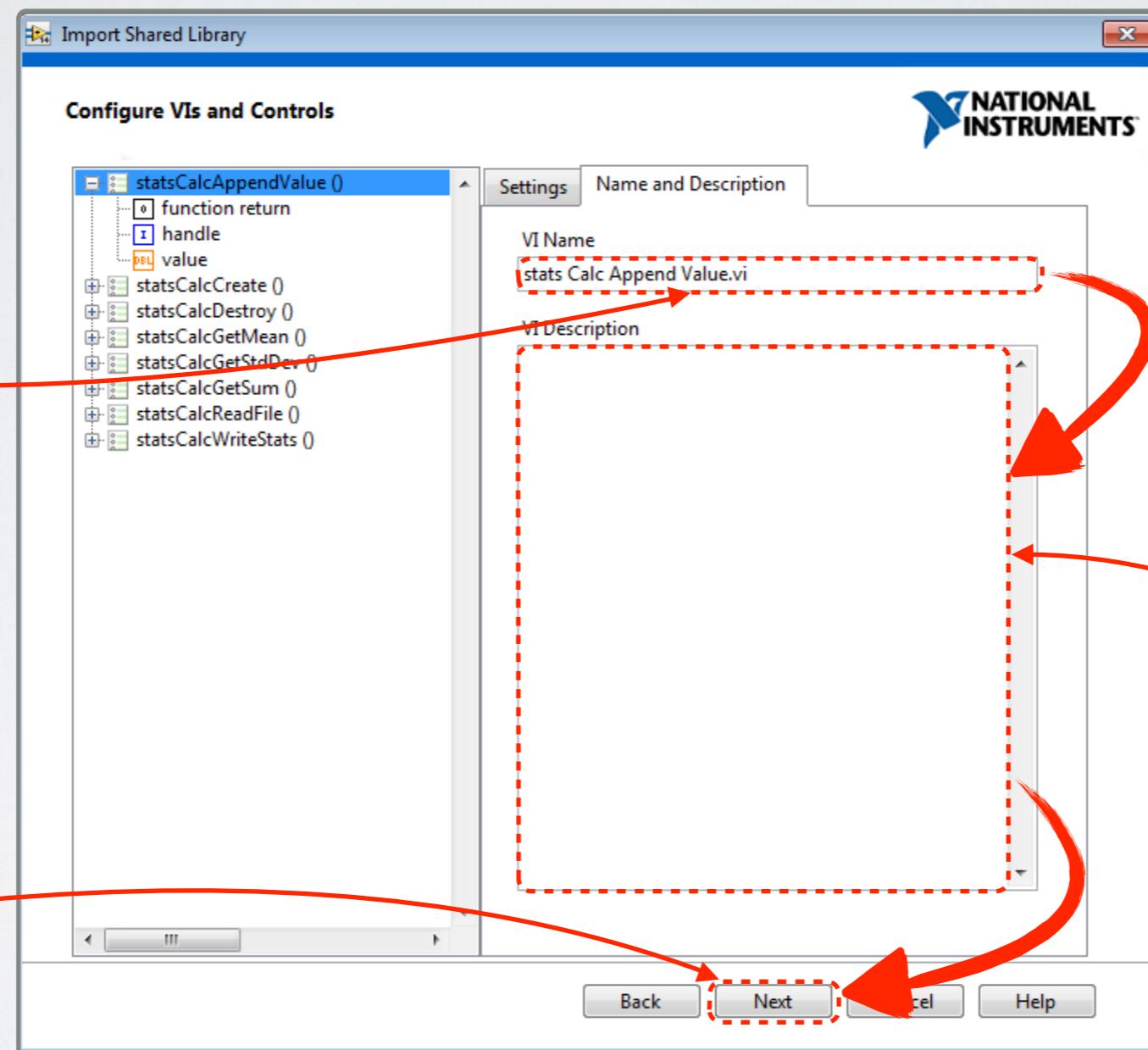


IMPORT SHARED LIBRARY

CONFIGURE VIS AND CONTROLS

Optionally specify a more user-friendly name for the VI that is mapped to the imported function.

Select “Next”



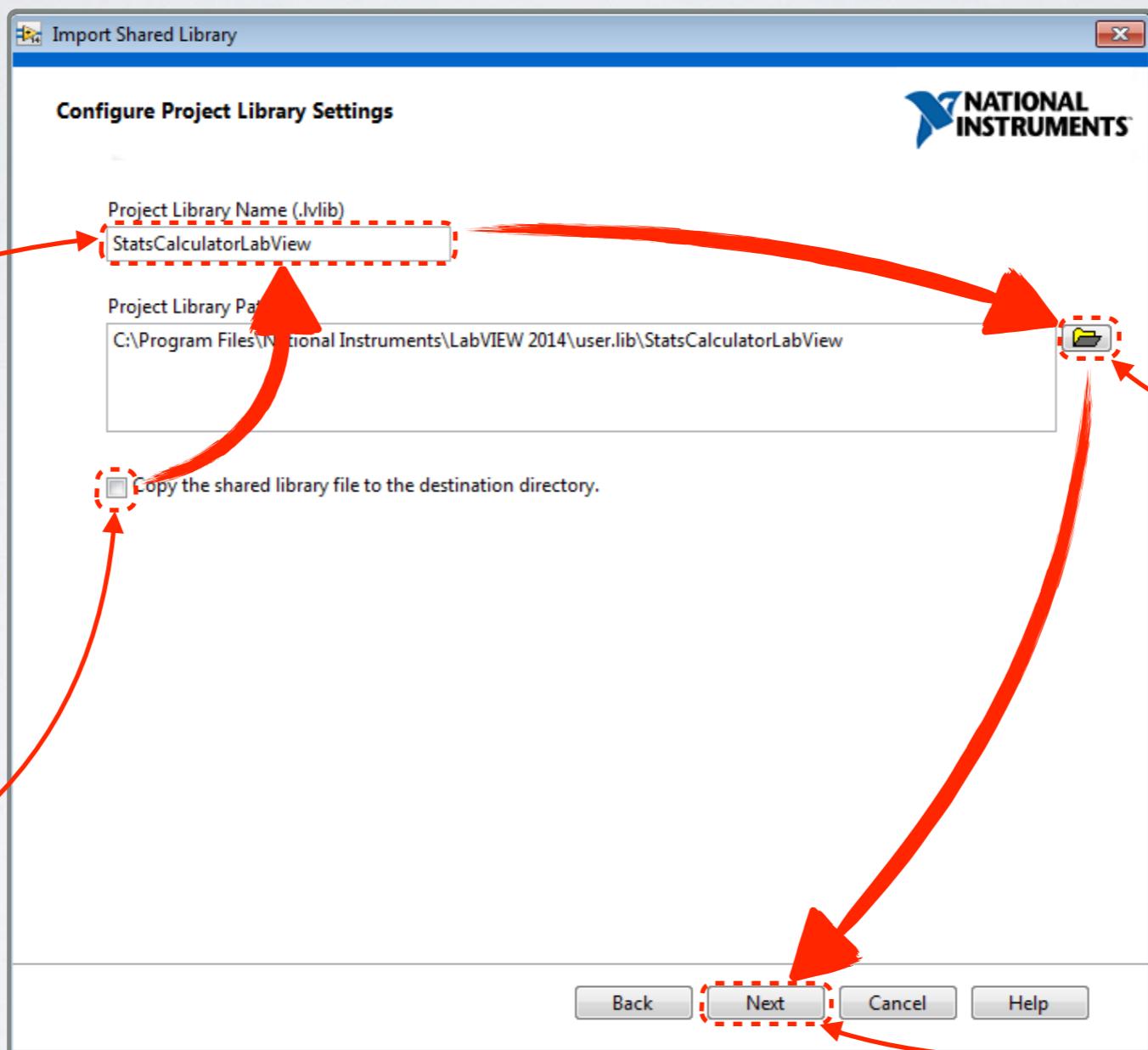
Optionally provide a description of the imported function's operation.

IMPORT SHARED LIBRARY

CONFIGURE PROJECT LIBRARY SETTINGS

Optionally alter the name of the LabVIEW VI library that will be generated.

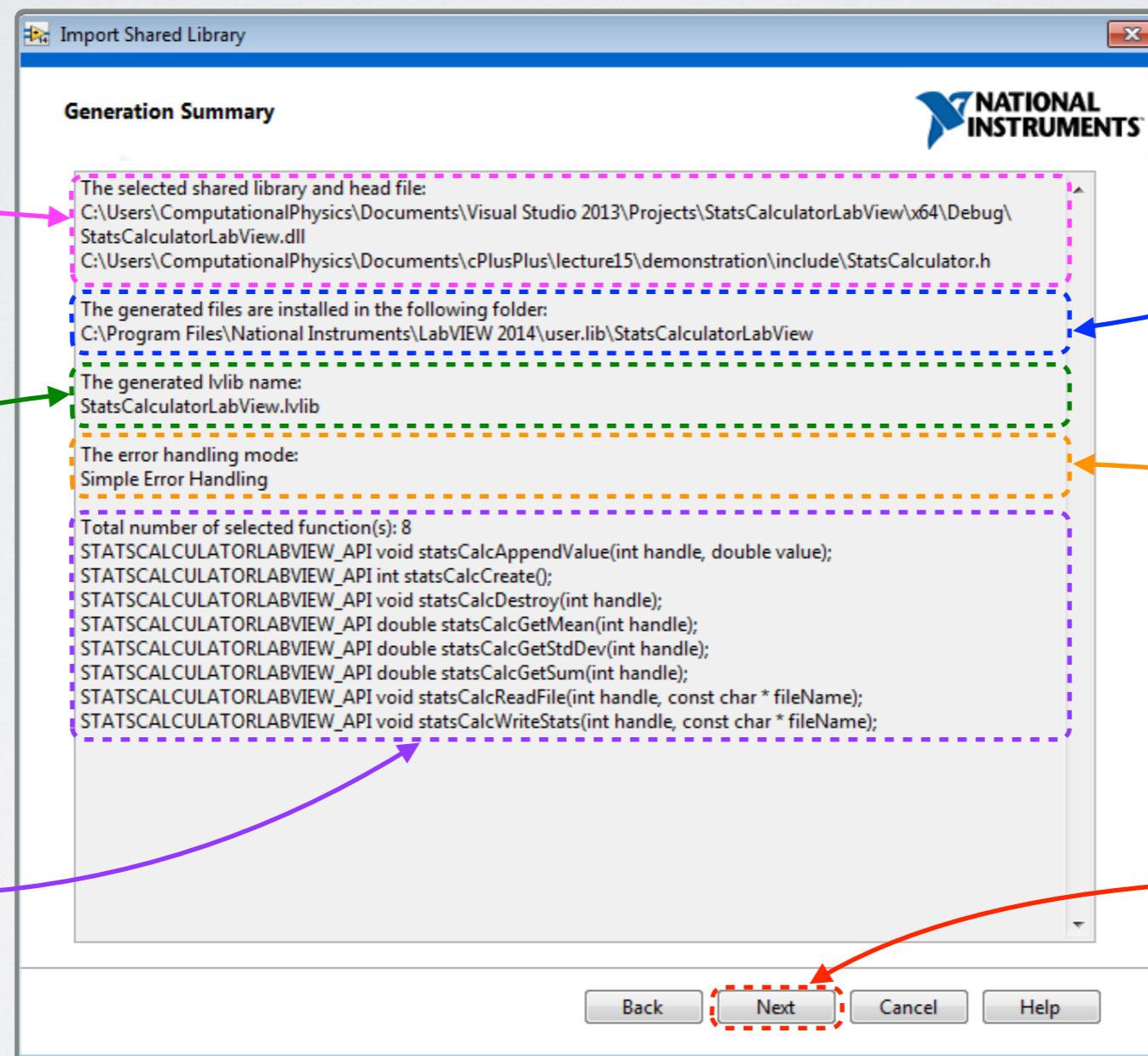
Verify that checkbox is **not** selected.



Optionally browse to select an alternative path for the LabVIEW VI library that will be generated.

Select "Next"

IMPORT SHARED LIBRARY GENERATION SUMMARY



Input file locations

Generated LabVIEW library name.

Imported C++ function list.

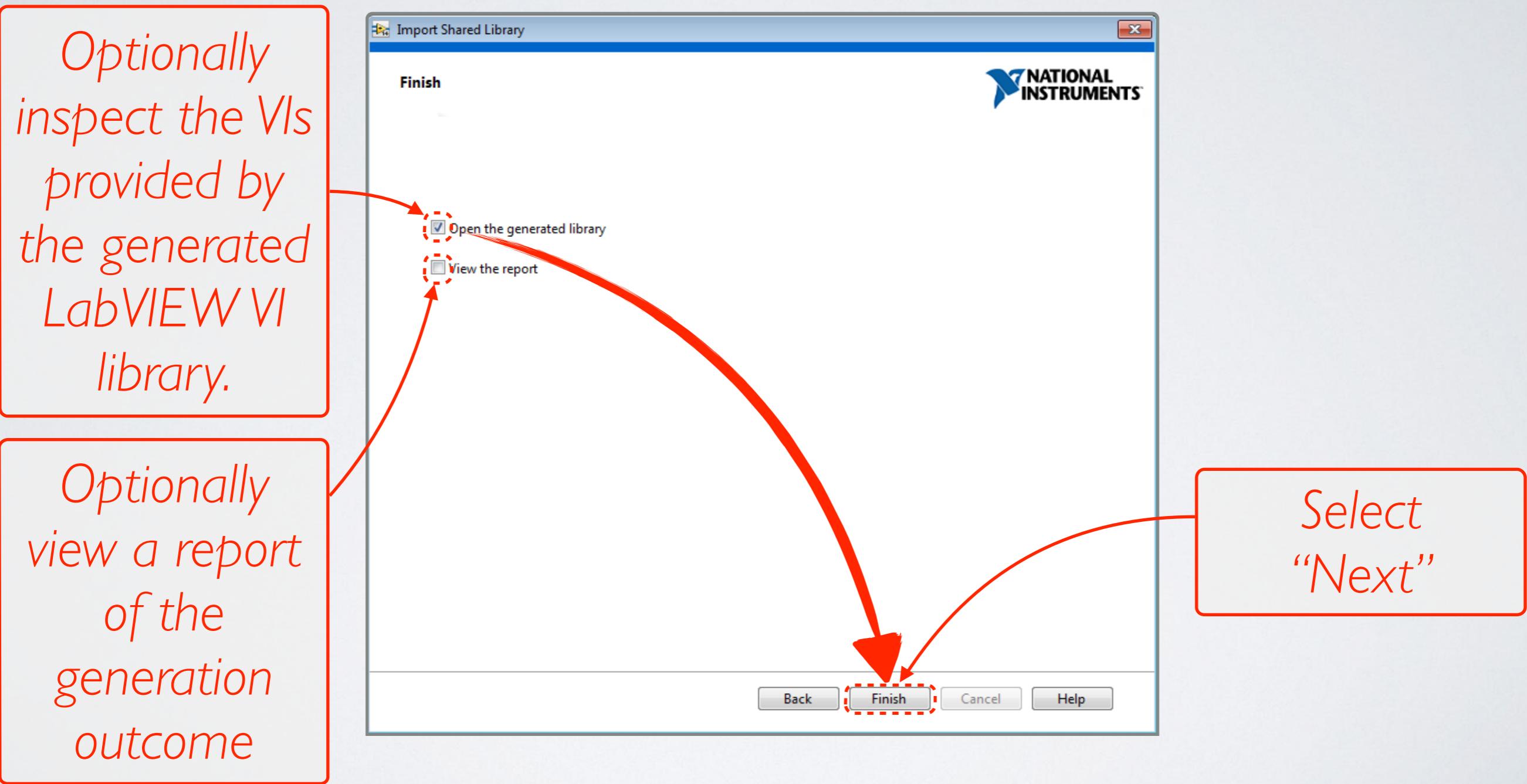
Generated files location.

Chosen error-handling mode.

Select “Next”

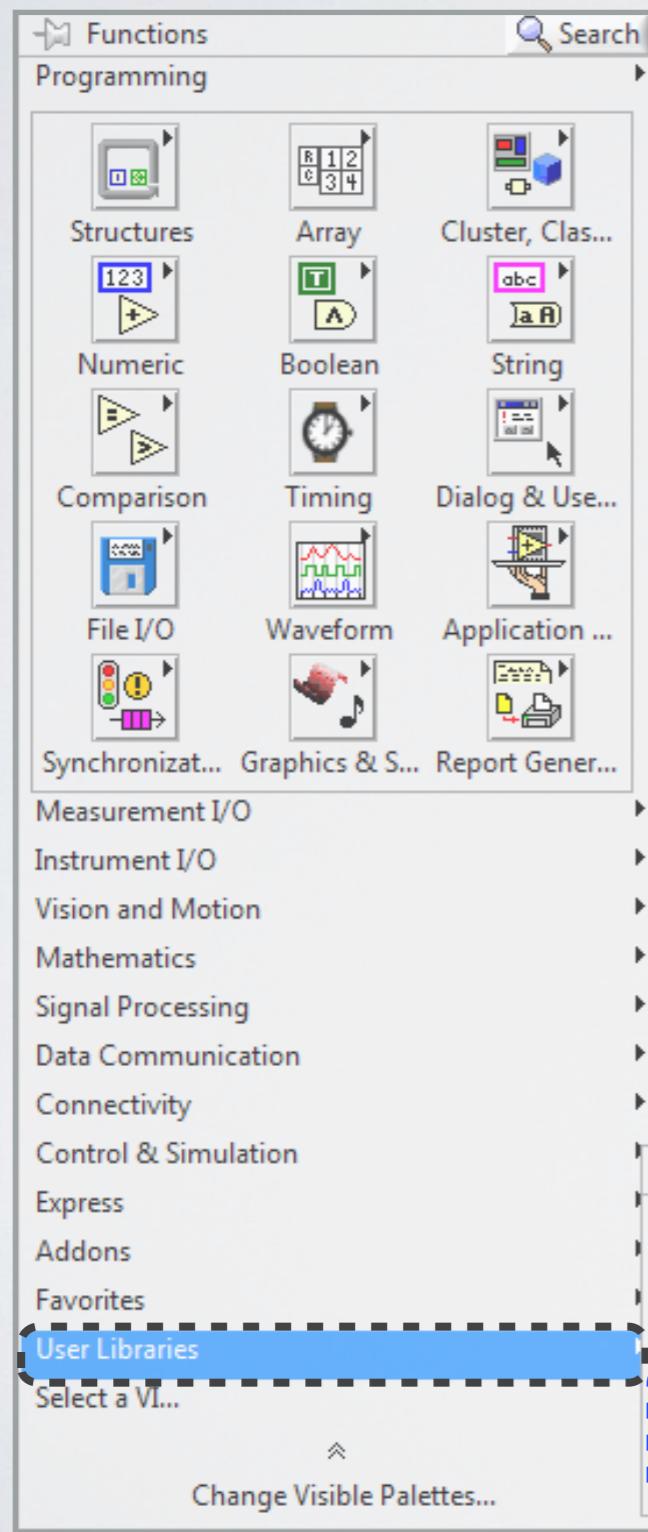
IMPORT SHARED LIBRARY

FINISH



STEP 7:
USE THE VIS

THE USER LIBRARIES PALETTE

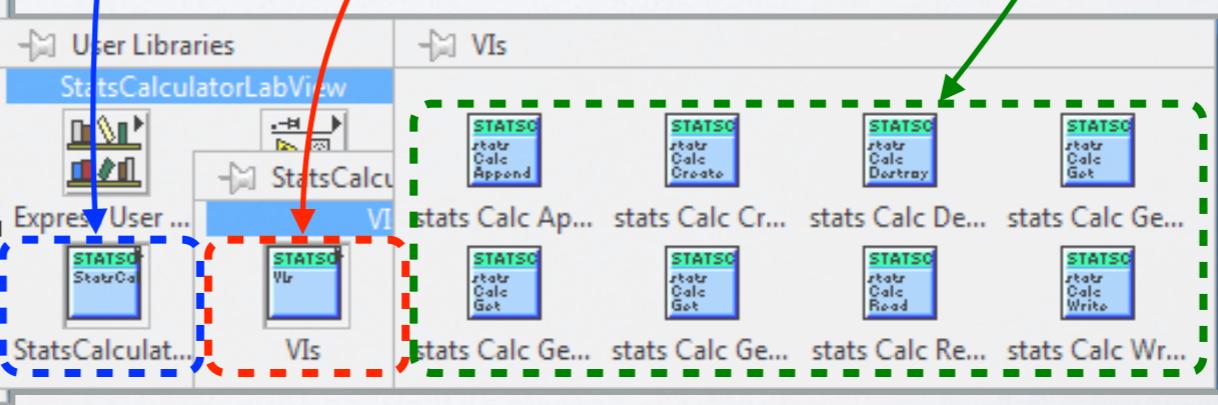


Palette for imported StatsCalculator functions

Only available from the **Block Diagram**

VIs have been generated for all imported functions

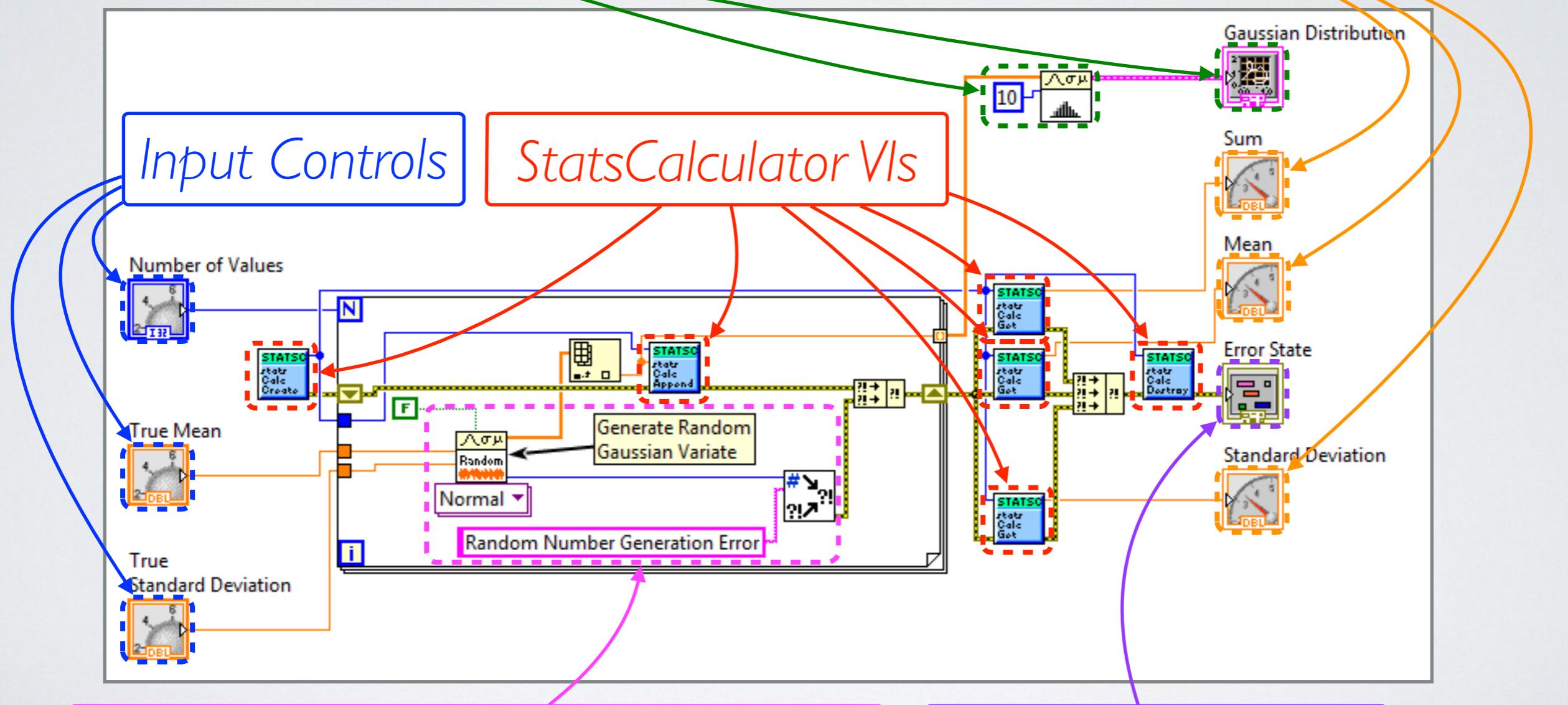
Sub-palette contains generated VIs



USING THE VIS BLOCK DIAGRAM

Histogram and display data

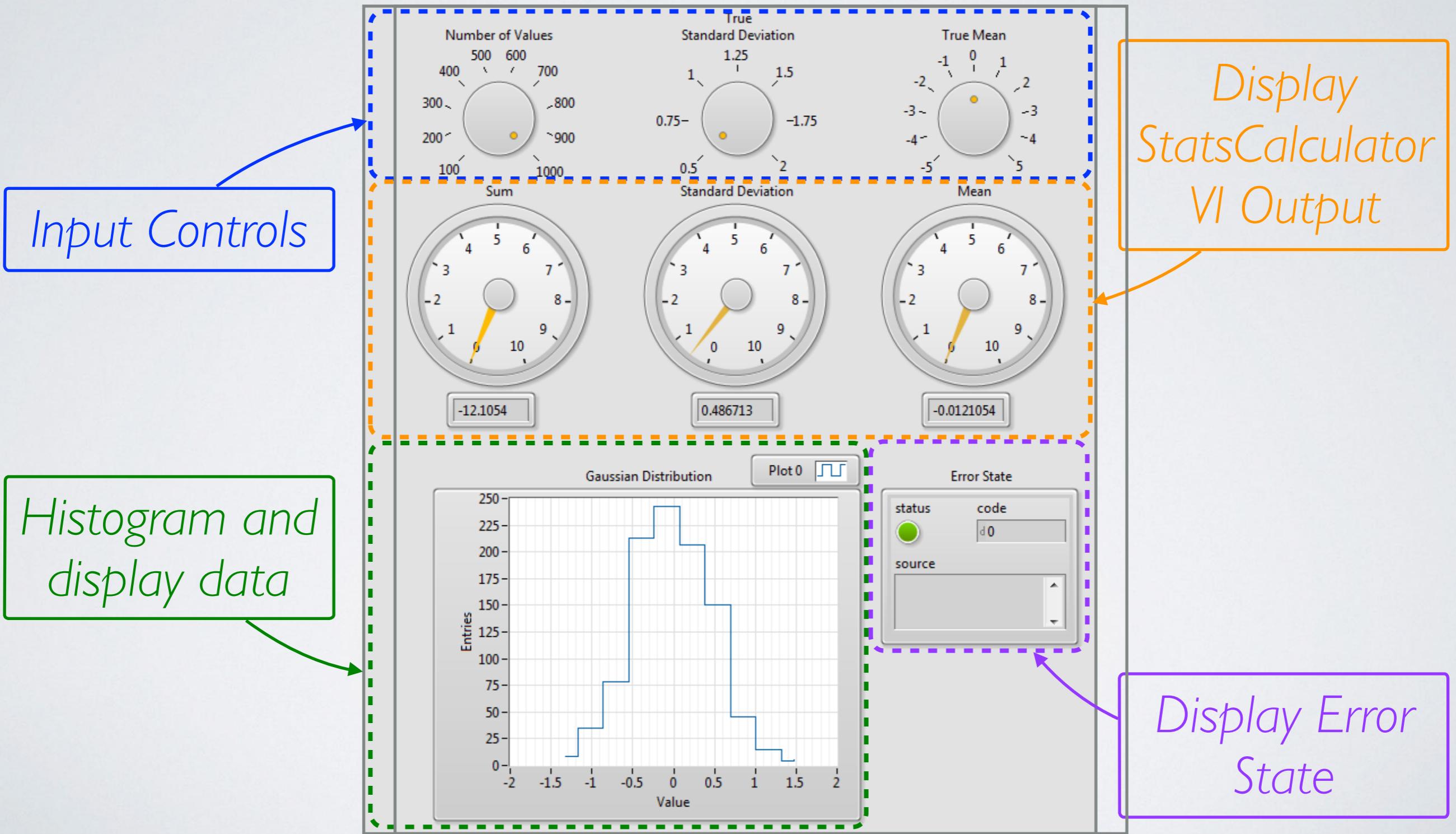
Display StatsCalculator VI Output



Generate Normal Random Variates

Display Error State

USING THE VIS FRONT PANEL



CREATING A NEW VI FROM THE BLOCK DIAGRAM

ENCAPSULATION OF FUNCTIONALITY

- When building **non-trivial** programs using LabVIEW, the **complexity** of the Block Diagram rapidly **increases**.
- Often, the Block Diagram can be meaningfully **subdivided** into **functionally distinct** subsets of elements.
- 👉 LabVIEW allows these subsets to be aggregated and saved as **new VIs**.
- In addition to simplifying the block diagram the newly created VIs can be **reused** in other LabVIEW programs.
- This is similar to the **encapsulation of functionality** provided by Object Orientated programming languages.

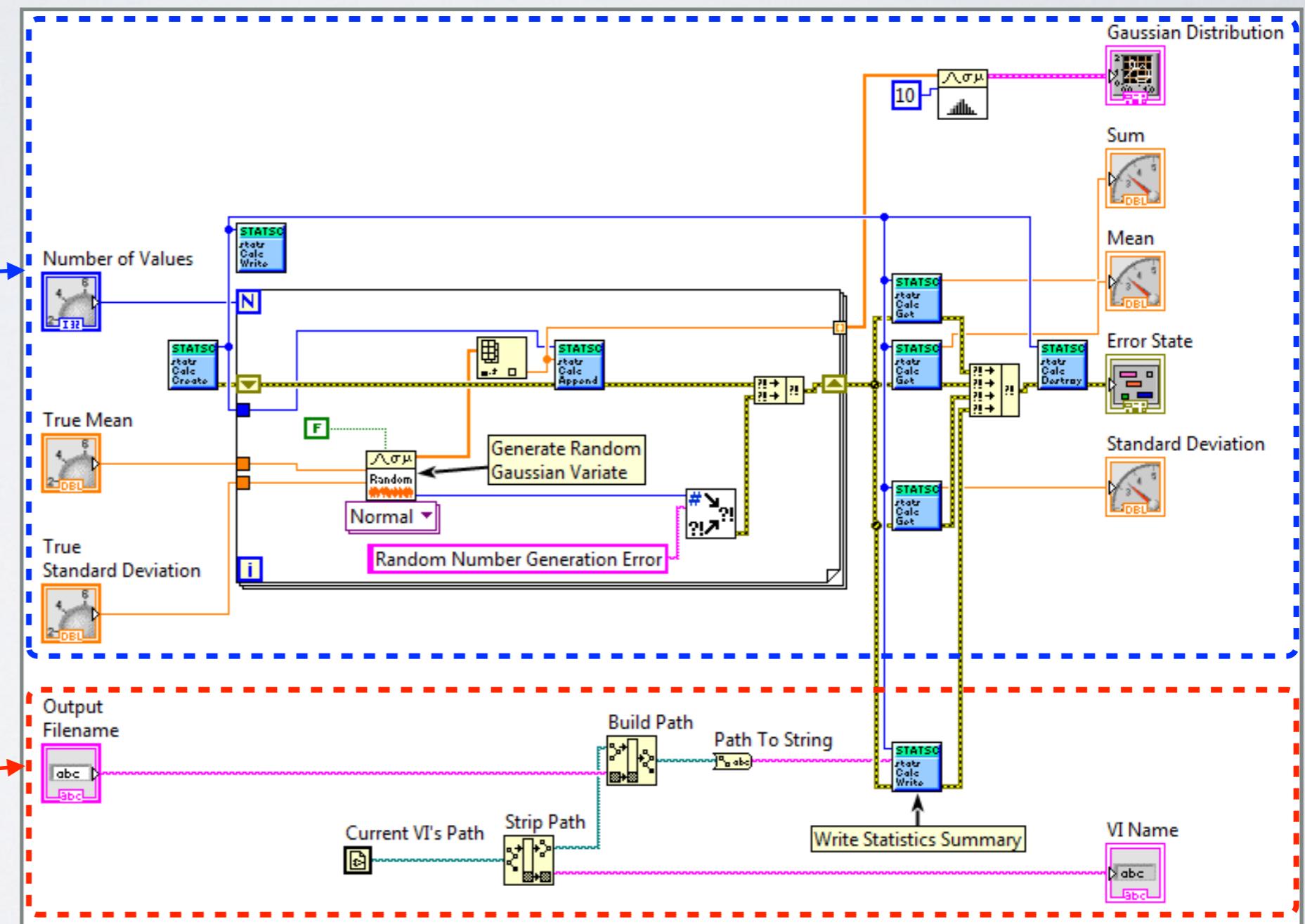
DEMONSTRATION

Creating a **New VI** from a **Subset of the Block Diagram**

CREATING A NEW VI IDENTIFY LOGICALLY DISTINCT CODE

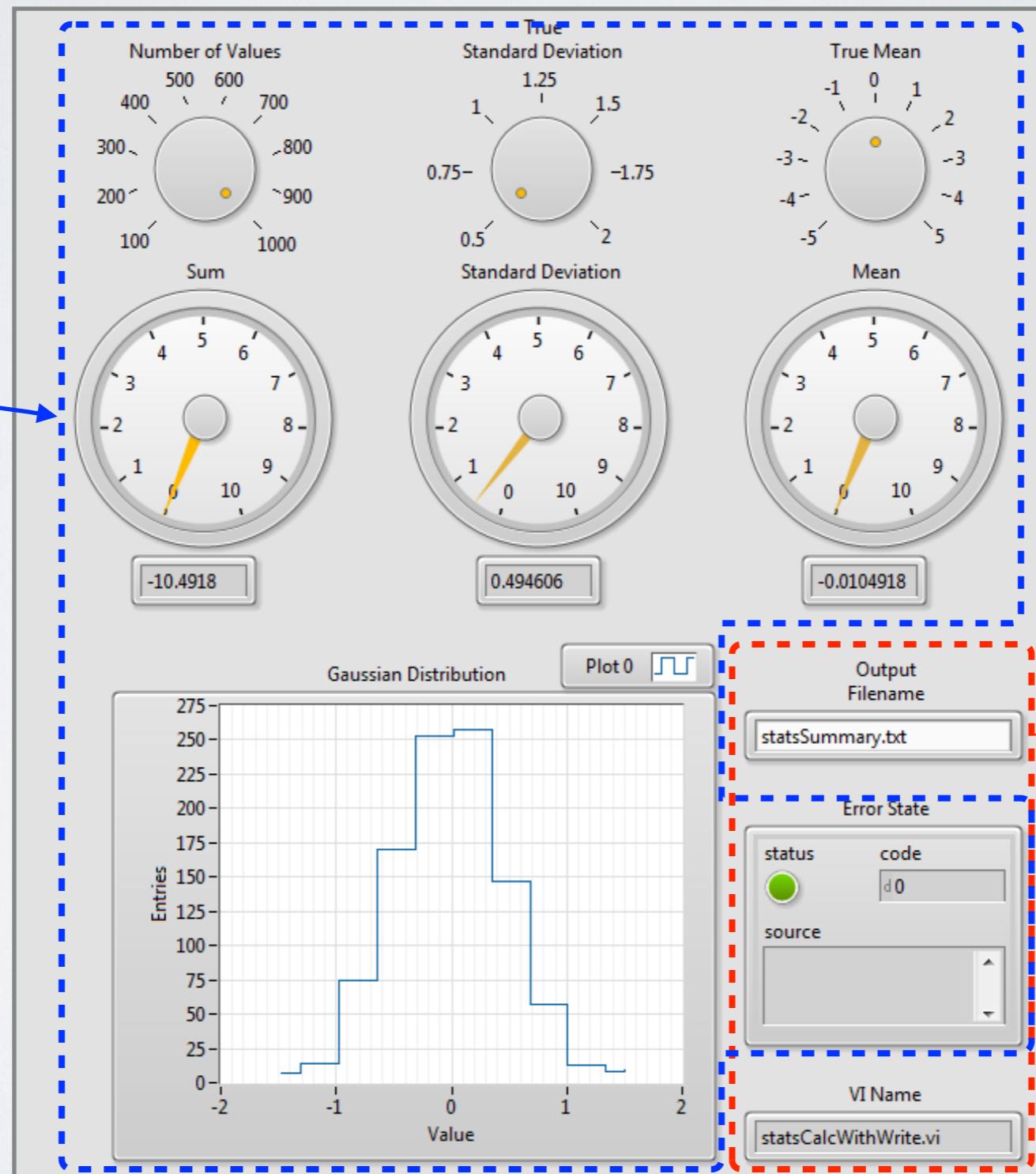
*Data generation,
analysis and
display logic.*

*File output and
path assembly
logic.*



CREATING A NEW VI

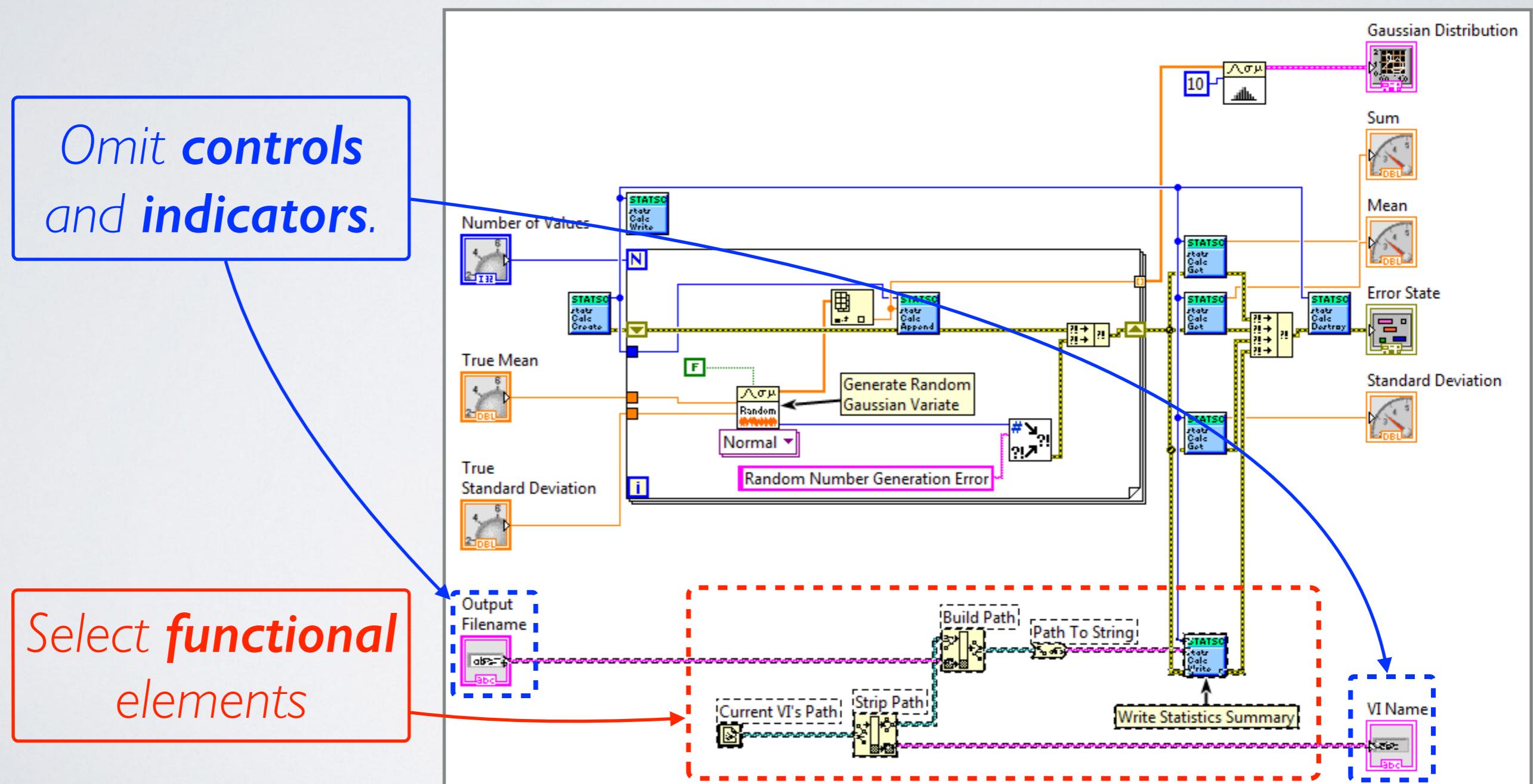
IDENTIFY LOGICALLY DISTINCT CODE



*File output and
path assembly
logic.*

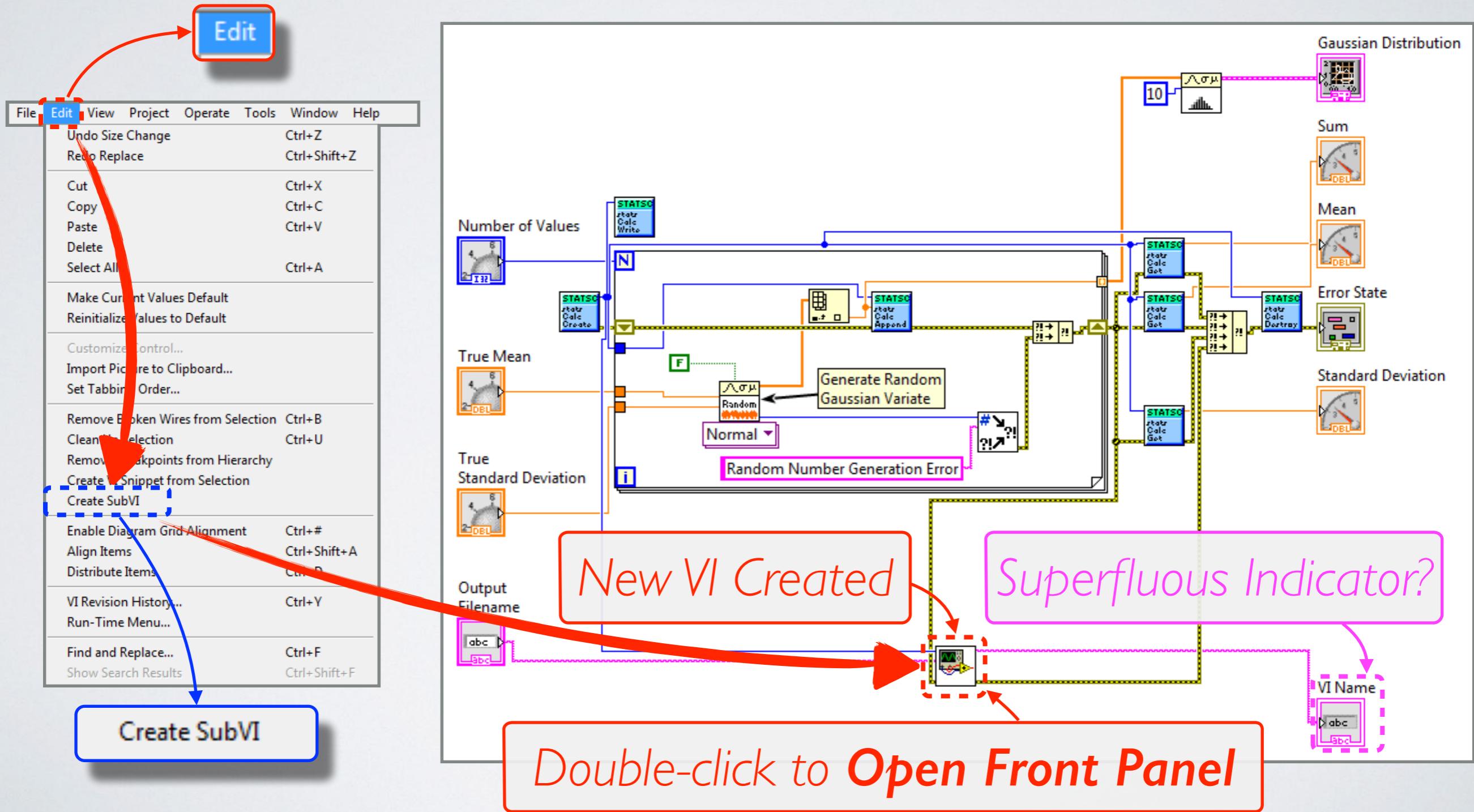
CREATING A NEW VI

SELECT BLOCK DIAGRAM ELEMENTS TO MERGE



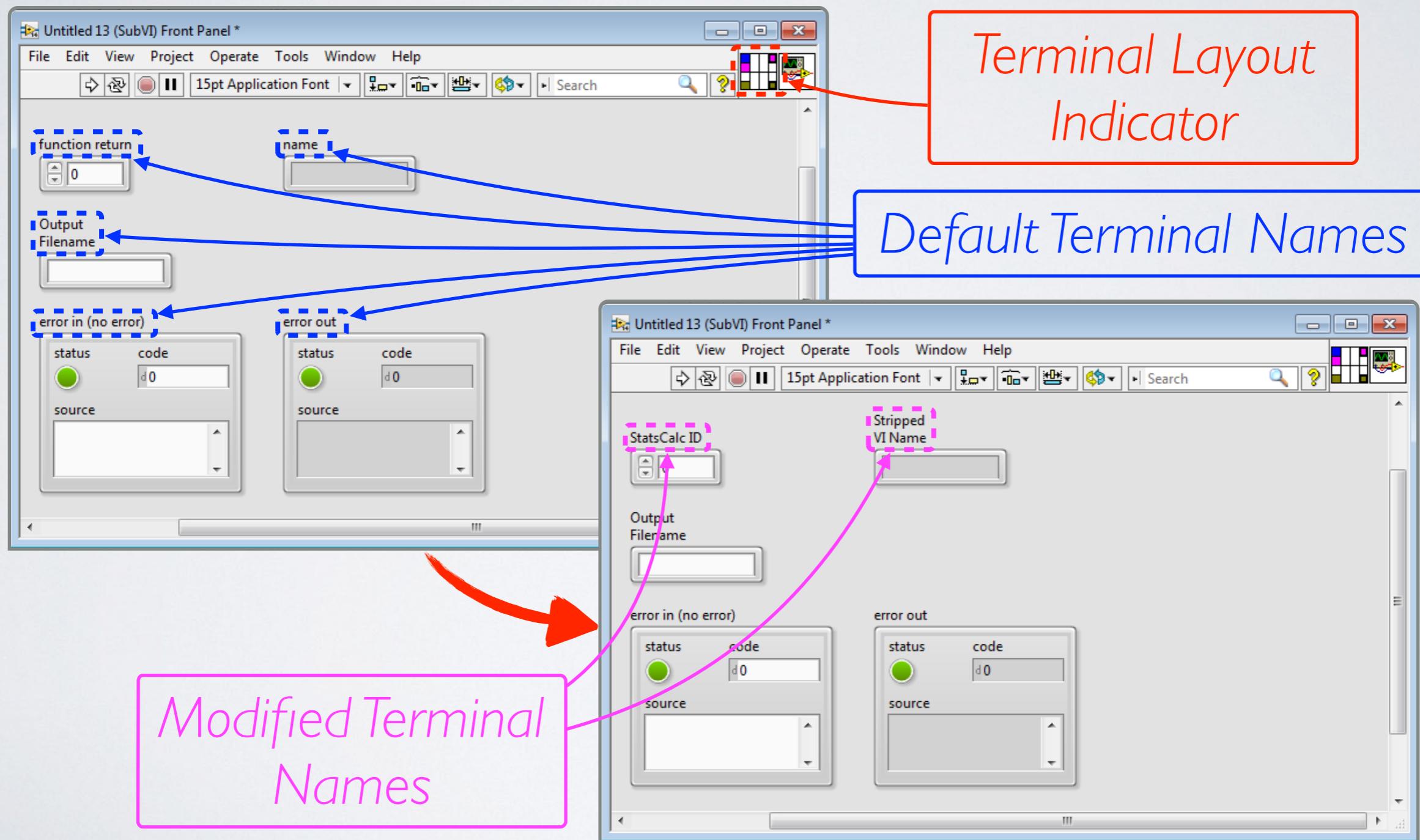
CREATING A NEW VI

SELECT *EDIT* → *CREATE SUBVI*



CREATING A NEW VI

MODIFY TERMINAL NAMES

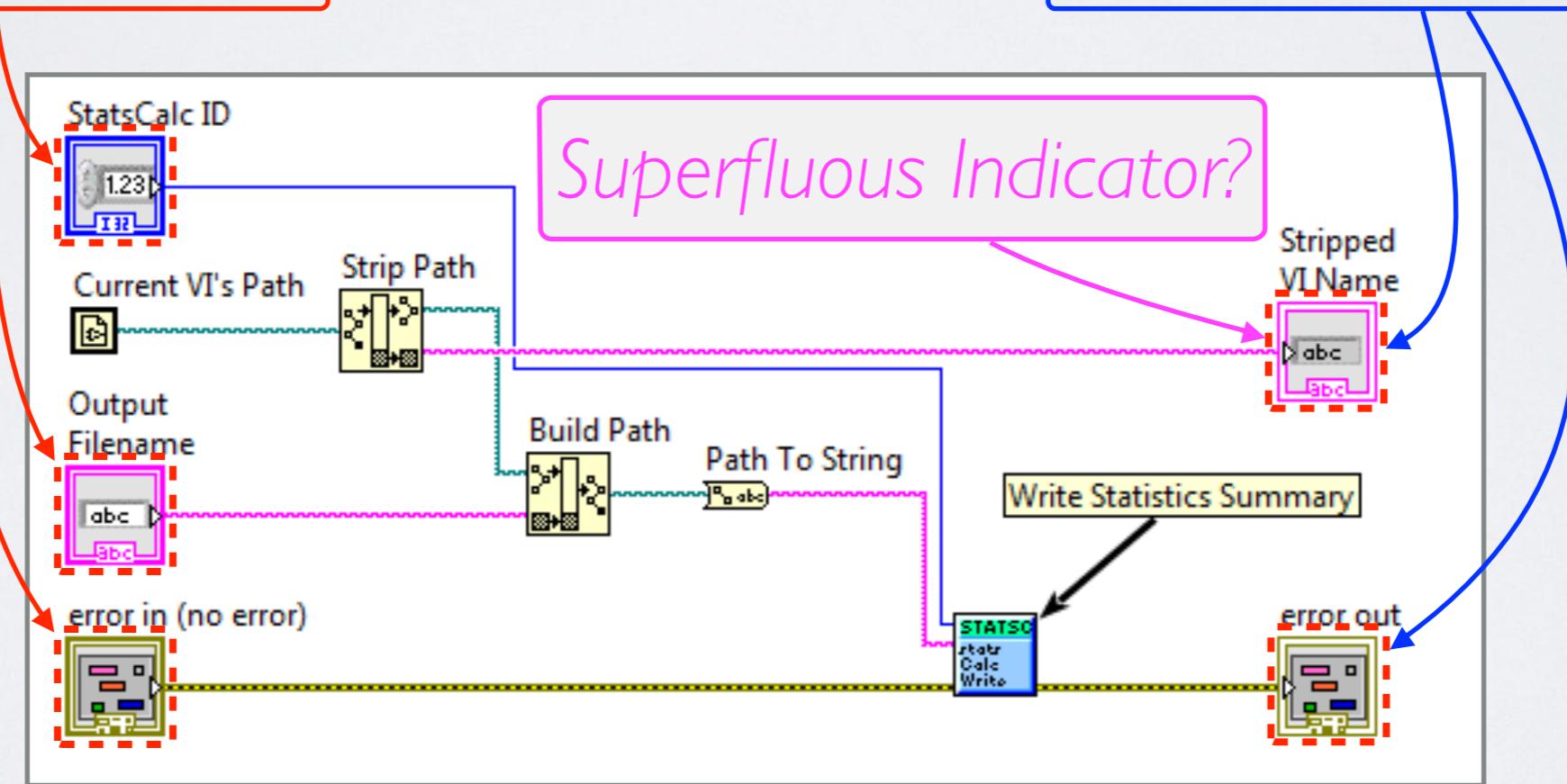


CREATING A NEW VI

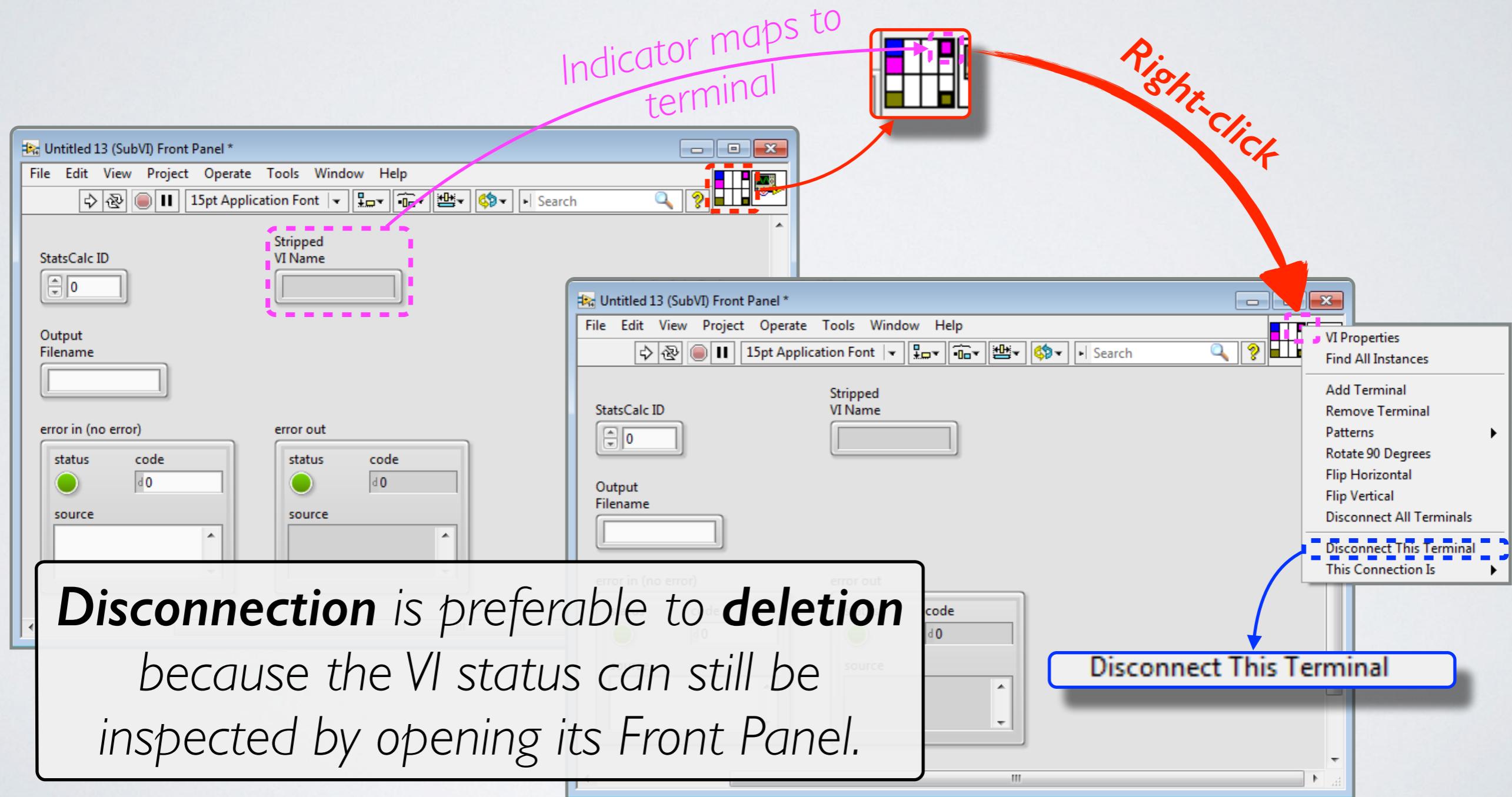
INSPECT NEW VI BLOCK DIAGRAM

Controls map to VI
input terminals

Indicators map to
VI **output** terminals

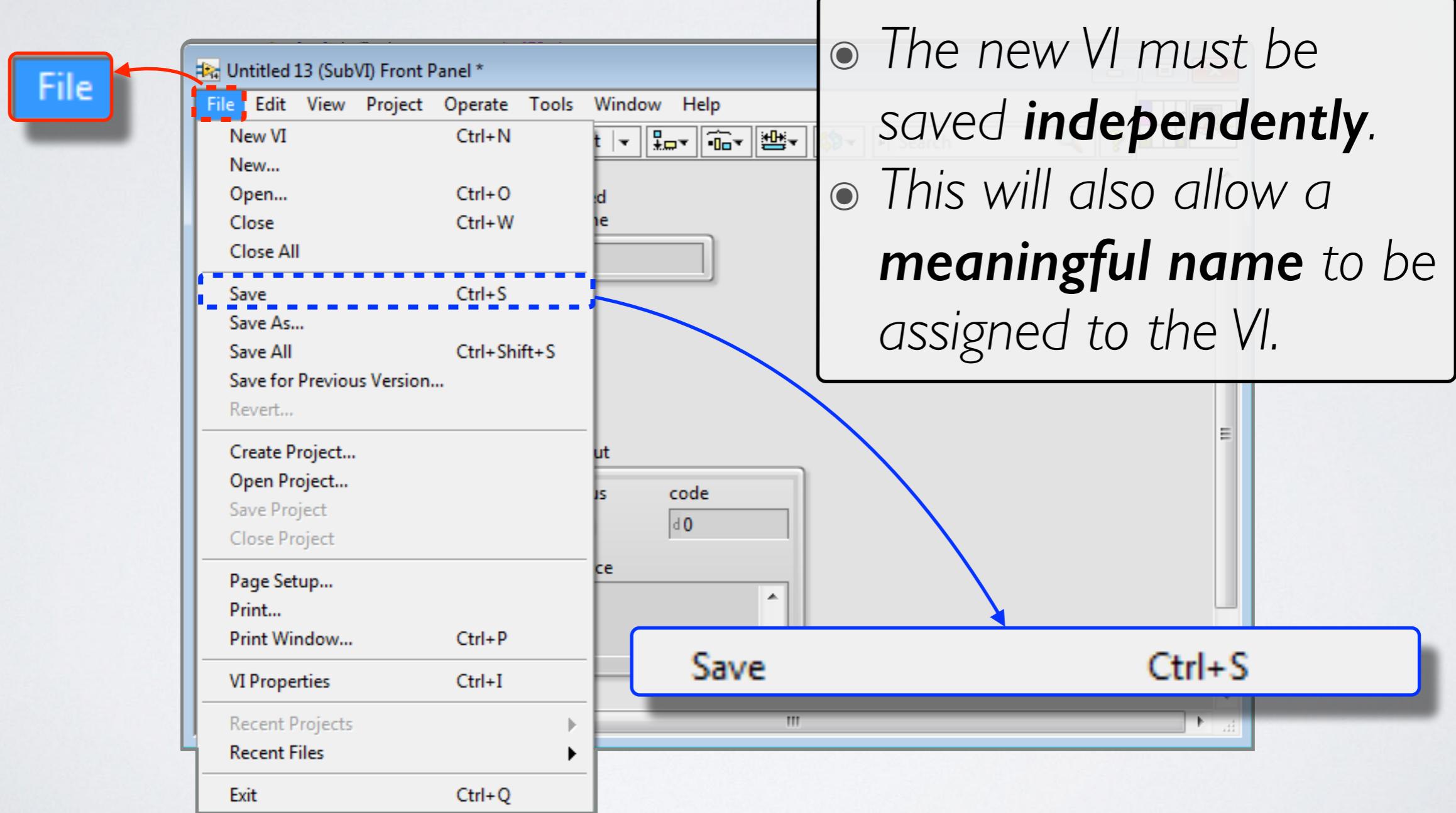


CREATING A NEW VI DISCONNECT SUPERFLUOUS TERMINAL



CREATING A NEW VI

SAVE THE NEW VI



RECOMMENDED READING

LabVIEW Manual: Creating SubVIs from Selections

(<http://zone.ni.com/reference/en-XX/help/371361L-01/lvhowto/creatingsubvisfromviselec/>)

LabVIEW Manual: System Exec VI

(http://zone.ni.com/reference/en-XX/help/371361L-01/glang/system_exec/)

LabVIEW Manual: Formula Node

(http://zone.ni.com/reference/en-XX/help/371361L-01/glang/formula_node/)

LabVIEW Manual: Using the Import Shared Library Wizard

(http://zone.ni.com/reference/en-XX/help/371361L-01/lvexcodeconcepts/importing_shared_library/)

LECTURE 15 HOMEWORK

Review the ***Recommended Reading*** items listed on the previous slide.

Continue to refine your **final project proposal** and continue working on your **final project**.