LA-UR-16-29620
Approved for public release; distribution is unlimited.

Title: SimApp 1.8x Release Notes

Author(s): Determan, John C.

Intended for: Release Notes for Software DIstribution

Issued: 2016-12-22

SimApp 1.8x Release Notes

Contents

The SimApp Source Code Release package consists of 3 zip files:

- ModelConverter.zip
- SimApp.zip
- SimAppUIATesting.zip

The file you are currently reading, SimApp Release Notes.pdf, is also included.

Installation

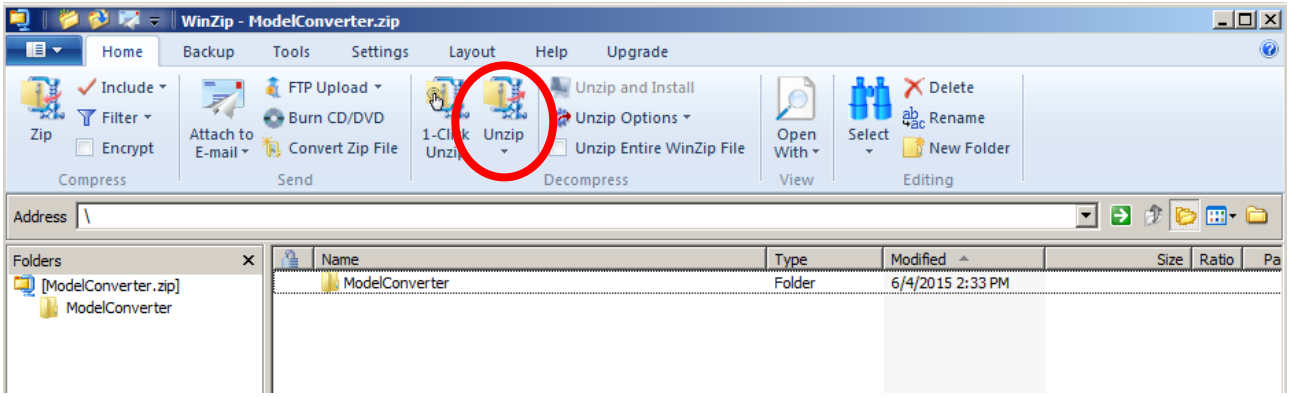
1. Extract the contents of each zip file to its own directory. While there are several options within WinZip for extracting files, the following procedure will give you full control of where to place the extracted files. Use the following steps on each zip file. These instructions are based on WinZip 15.

    a. Open the selected zip file (double click on icon).
    b. Locate **Home \ Unzip** in the ribbon and click lower half of button to reveal menu.
    c. Select "Set default unzip folder" – a dialog opens.
    d. Choose folder. The top level folder from the zip file will be created under the selected location.
    e. Click again on **Home \ Unzip \ Set default unzip folder** – your selected folder is in the list, select it. The contents of the zip file will be extracted to the selected location.

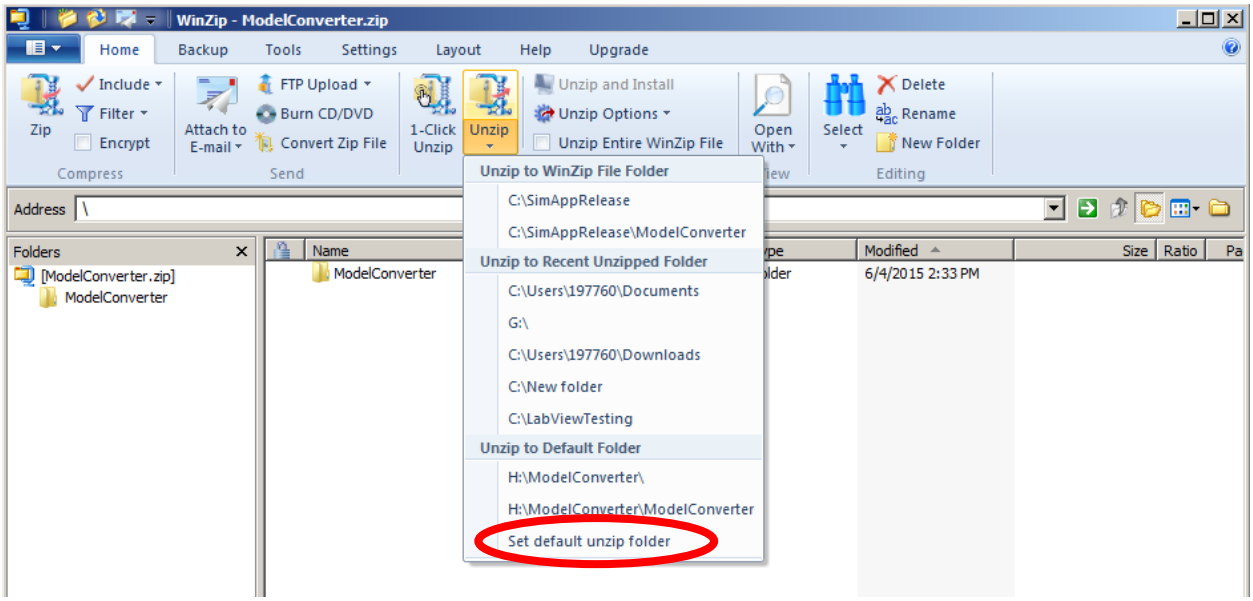
Example

The extraction procedure outlined above will be illustrated on the ModelConverter.zip file.

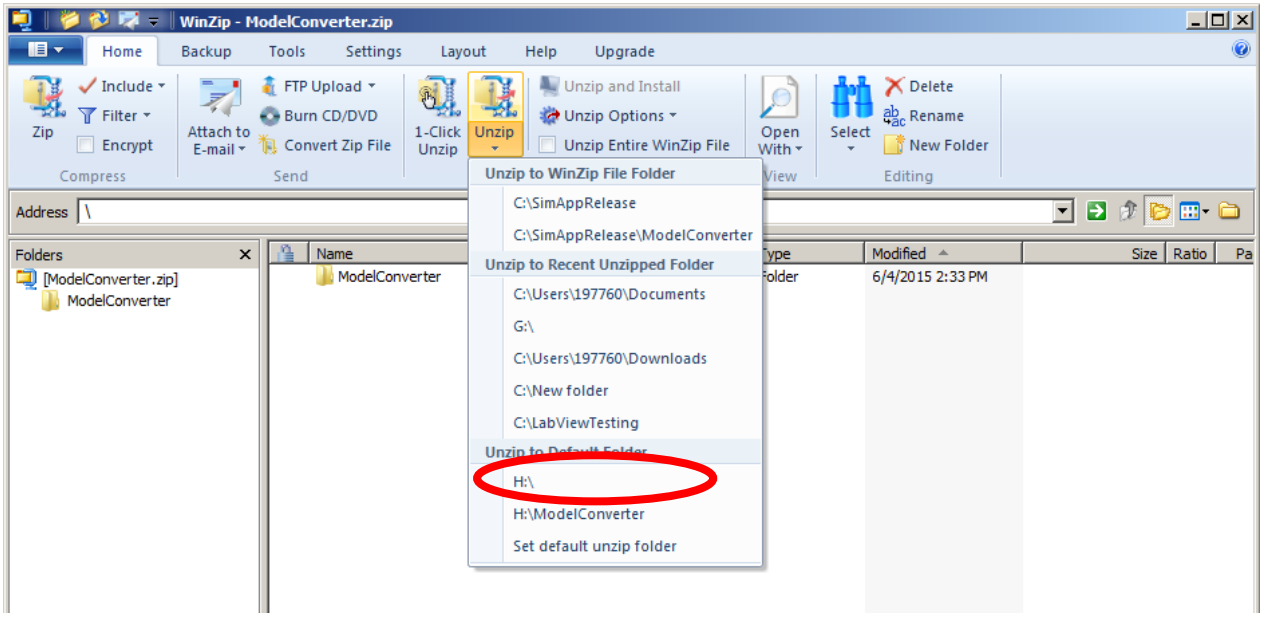
Upon opening the ModelConverter .zip file the user will see:



The ribbon is at the "Home" tab; the "Unzip" button is circled. Click the lower half of the button, the word "Unzip" to reveal the menu, as shown below. The "Set default unzip folder" is circled in red.



This example will select Drive H: as the default location. The Unzip menu will then look as shown below. The default just set is circled in red. Select that location, and the top level folder "ModelConverter" will be created on Drive H.



Compilation

Visual Studio uses a hierarchy to represent source code. The top level is called a "Solution". Solutions contain "Projects". Projects contain files, which may be further subdivided, depending on the language being compiled. C++ projects will typically have folders for header files, for source files and for resource files. C# projects will not separate the files in that manner. The Solutions in this release have been

compiled using Microsoft Visual Studio 2013. Use of a later version of visual studio will automatically update the project files to that version of Visual Studio. Compilers can change between versions, so some (usually minor) changes may be required to the code.

The general procedure for compiling a Solution in Visual Studio will be presented below. The steps are based on use of Visual Studio 2013. Working with each specific Solution will be detailed the remaining sections.

1. Open the solution file.
   a. Use the menu bar to select **File \ Open \ Project/Solution…** .
   b. A dialog opens. Select the drive and folder of the solution to be opened. The folder contents pane of the dialog will include a file name of the form "*.sln" where * is the name of the solution to be opened. Select this file and click "Open".
   c. The projects in this release were developed under source control, so a dialog asking if you wish to use source control will open. Select "No". A second dialog will open concerning source control. Select "OK". Source control was managed using Microsoft Team Foundation Server (TFS). Use of TFS is beyond the scope of this document, as different organization will handle source control in different ways.
   d. The solution will now be open. Use the "Solution Explorer" pane to browse the contents of the solution. If the Solution Explorer pane is not open, click **VIEW \ Solution Explorer** in the menu bar. The Solution Explorer pane will open.

2. Compilation is handled with the **BUILD** menu items. The projects of this release contain two types of project configurations, "Debug" and "Release". Usually the first step in compilation is to select a configuration. Click **Build \ Configuration Manager…** . A dialog opens. From the "Active solution configuration" menu select either Release or Debug. Click the "Close" button.

3. The user has many options at this point. One option is **BUILD \ Clean Solution**. This will remove files that have been created by the compiler, such as object files, executable files, or any other intermediate files. This option will force all files to be recompiled, and all project components to be relinked. This will not be done frequently, but is one way to know that only the most recent source code is being used. Another option is **BUILD \ Rebuild Solution**. This option will automatically perform the clean step just described, and then recompile and relink all components from source. The option **BUILD \ Build Solution** is also available. The idea here is to save some compilation time by retaining old object files and only recompiling those source files that have changed. The author tends to use **BUILD \ Rebuild Solution** as the compile time is not long for the solutions in this release, and the results of the operation are absolutely certain – only the correct code has been used. Other build options will not be discussed in this document.
4. The **PROJECT** menu also has some items that are good to be aware of for initial compilation of a solution. Appropriate options will already be saved in the solution files, so no actions described in this section should be needed, but these are good points to be aware of.

   a. A solution containing multiple projects needs to know several things about the relationships among the projects. One and only one of projects will be the "Startup Project". This project

will typically be an application with a graphical user interface, but could be a console application as well. The Startup Project will be an executable, as opposed to a library of some sort. The Startup Project is specified by choosing a project in the Solution Explorer and then clicking **PROJECT \ Set as Startup Project** in the menu bar. Projects are the first level entries under the Solution name; select a project by clicking on the project name. The Startup Project name is formatted in bold letters, the other project names are not.

b. Some projects depend on other projects – a project that uses resources from a second project is said to *depend* on that second project. This means that dependencies and build order have to specified. The menu items **PROJECT \ Project Dependencies…** and **PROJECT \ Project Build Order…** both open the same dialog where tabs select between these options. Specifications are made under the Dependencies tab. Dependencies are specified by selecting a project name in the Projects drop down menu and checking the names of other projects on which the selected project depends. Specify dependencies for each project that depends on another project. Once all dependencies have been specified the build order of the projects has been determined and is displayed in the Build Order tab.

c. The remaining **PROJECT** option of interest is **PROJECT \ Properties**. Clicking on this option opens a window where all project properties can be specified. These properties include compile options, link options, custom build options and more. While the solutions in this release should compile under Visual Studio 2013 with the currently selected options, use of any more recent version of Visual Studio may require specification of some options through the property pages referenced here.

5. Use Windows Explorer to examine the results of successful compilation. The compiled and linked executable will exist in the appropriate folder, as described below. The Solution will be the top level folder.

   a. If a solution contains more than one project, a folder for each project exists in the solution folder. In addition, there are Release and Debug folders in the Solution folder. The final products of compilation end up in these folders – the executable file and any libraries on which the executable depends. The Release folder holds the results of compiling the release configuration of the Solution and the Debug folder holds the corresponding compilation products for the debug configuration.

   b. If a solution contains only a single project, there will neither be Release nor Debug folders at the solution level. Instead, the project folder will contain a “bin” folder and the Release and Debug folders will be contained in bin. The desired executable file appropriate to the configuration will be in the Release or Debug folder.

6. Run the executable code under Visual Studio.

   a. To run the release version, compile the release configuration as described above.

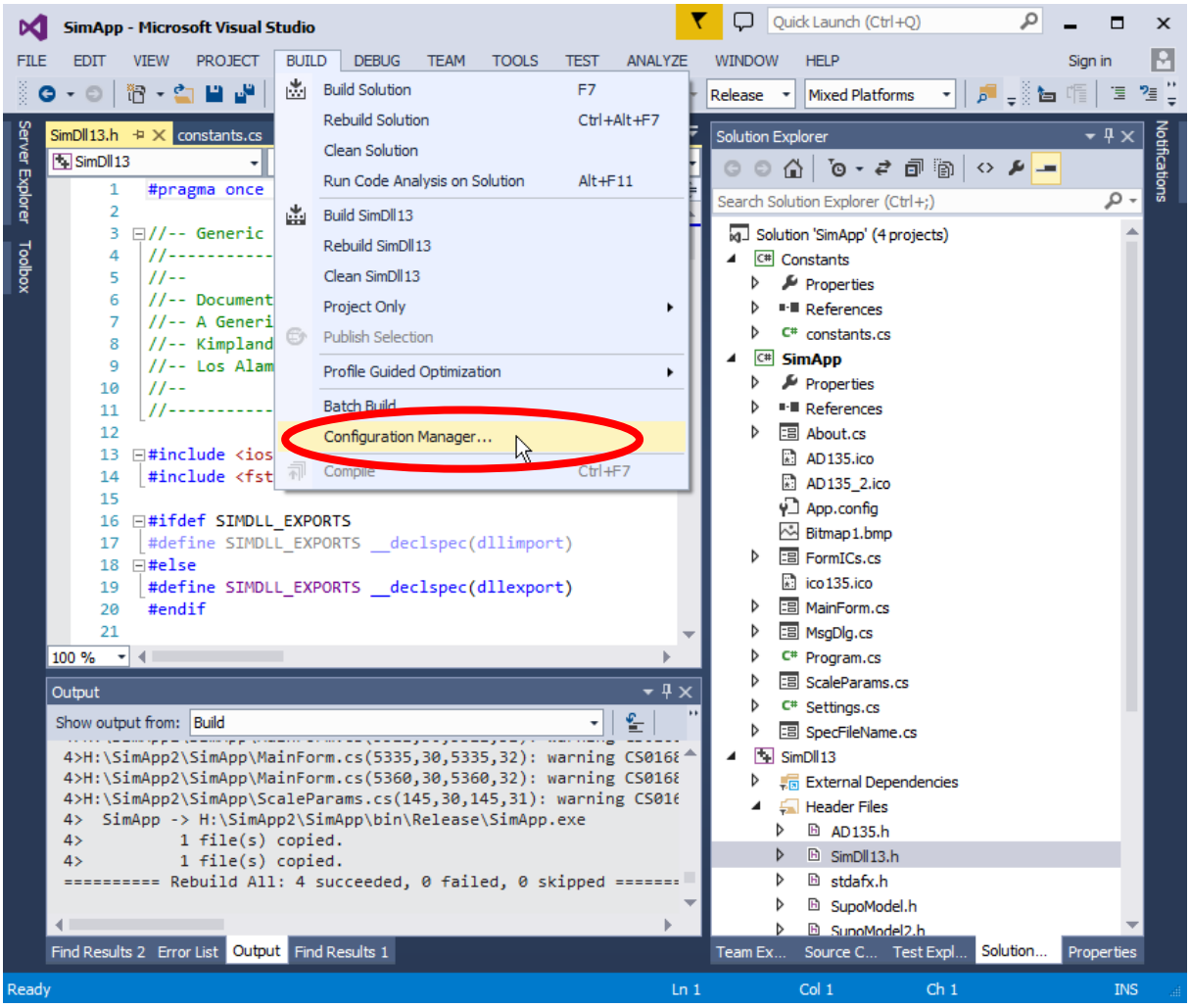
b. Click **Debug \ Start Without Debugging** – the application will open and run in release mode. Typically much faster than debug mode, but of limited help in fixing problems.

c. To run the debug version, compile the debug configuration as described above.

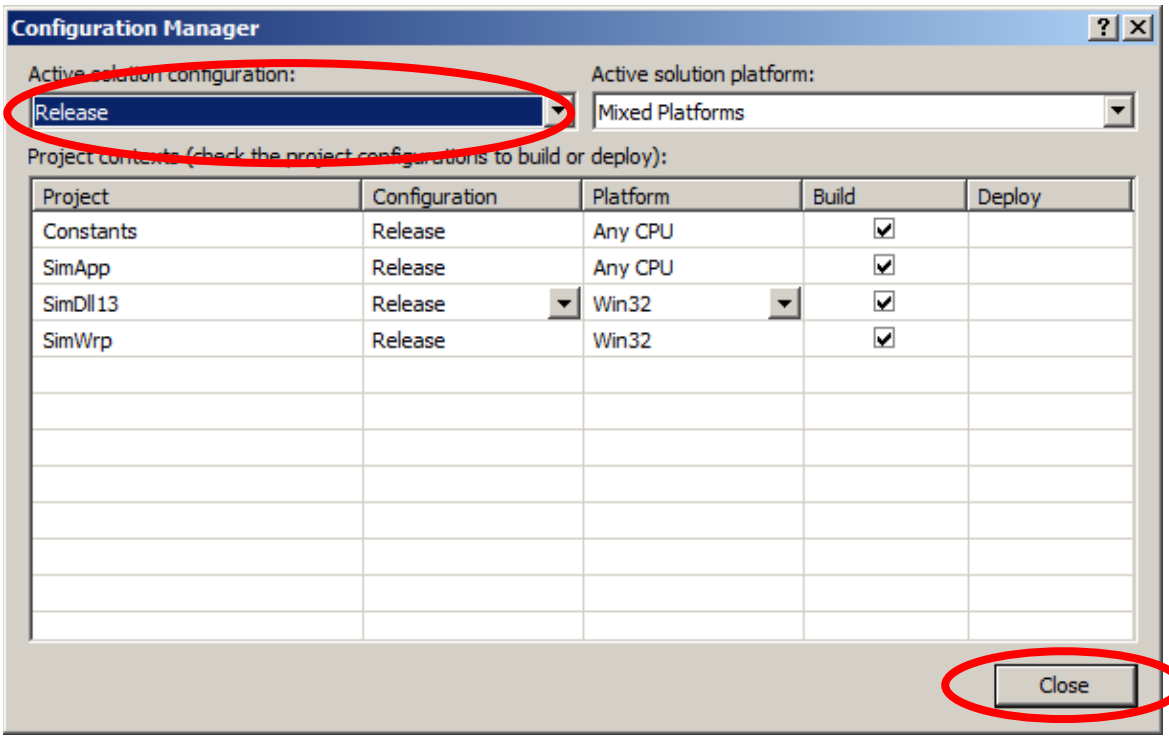
d. Click **Debug \ Start Debugging** – the application will open and run in debug mode. Typically much slower than release mode, but allows setting breakpoints, catching errors, etc.

Example

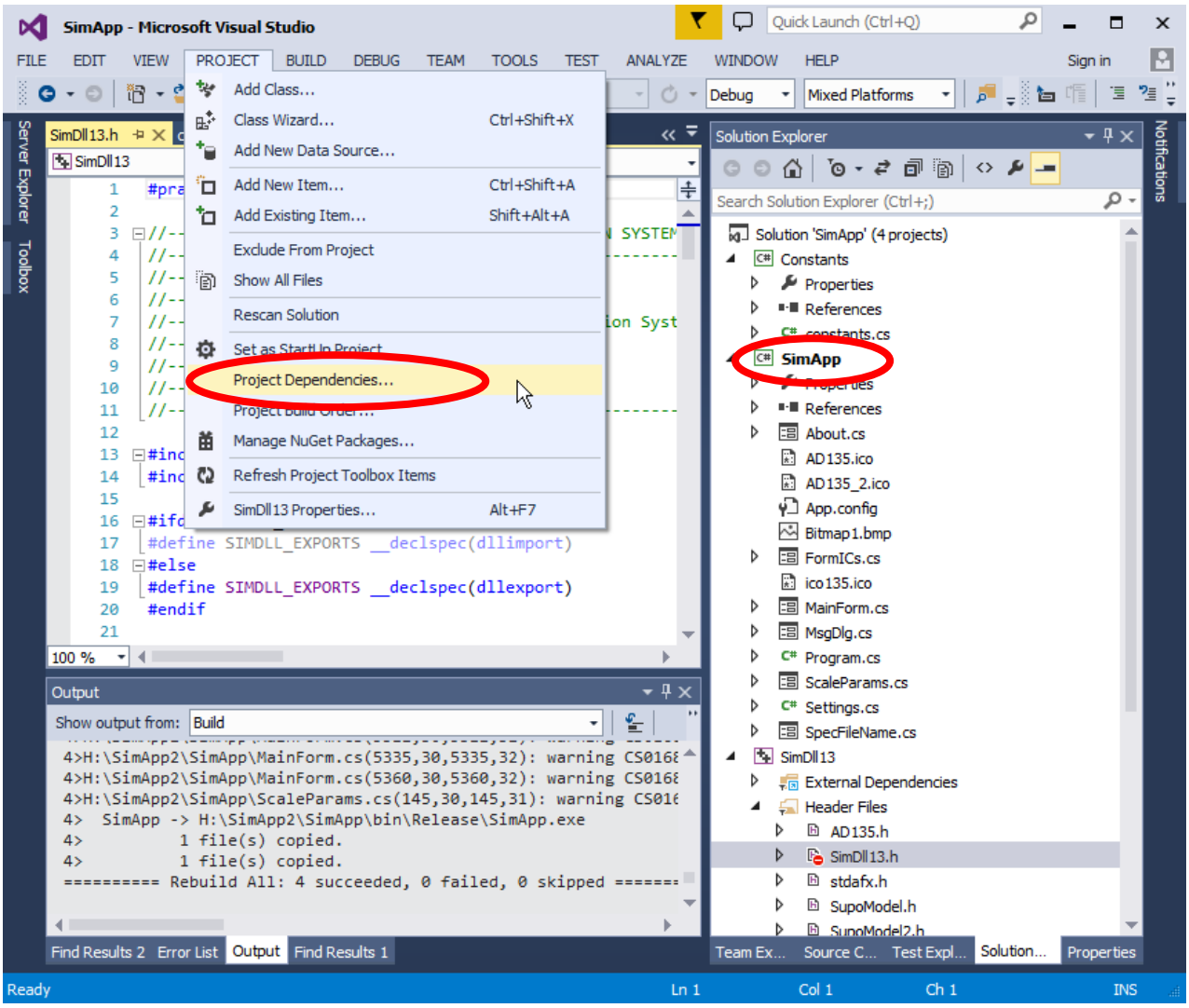
Compilation will be illustrated for the solution SimApp. Open Visual Studio, and use the **FILE \ Open \ Project/Solution…** process to open the SimApp Solution installed earlier. With SimApp open, click **BUILD** in the menu bar and locate **Configuration Manager…** . The figure below shows this step. Note the typical configuration of a code pane, a Solution Explorer pane, and an Output pane.



Click **Configuration Manager…** and the following dialog opens. Select the Debug configuration from the menu circled in red on the upper left, then click the “Close” button

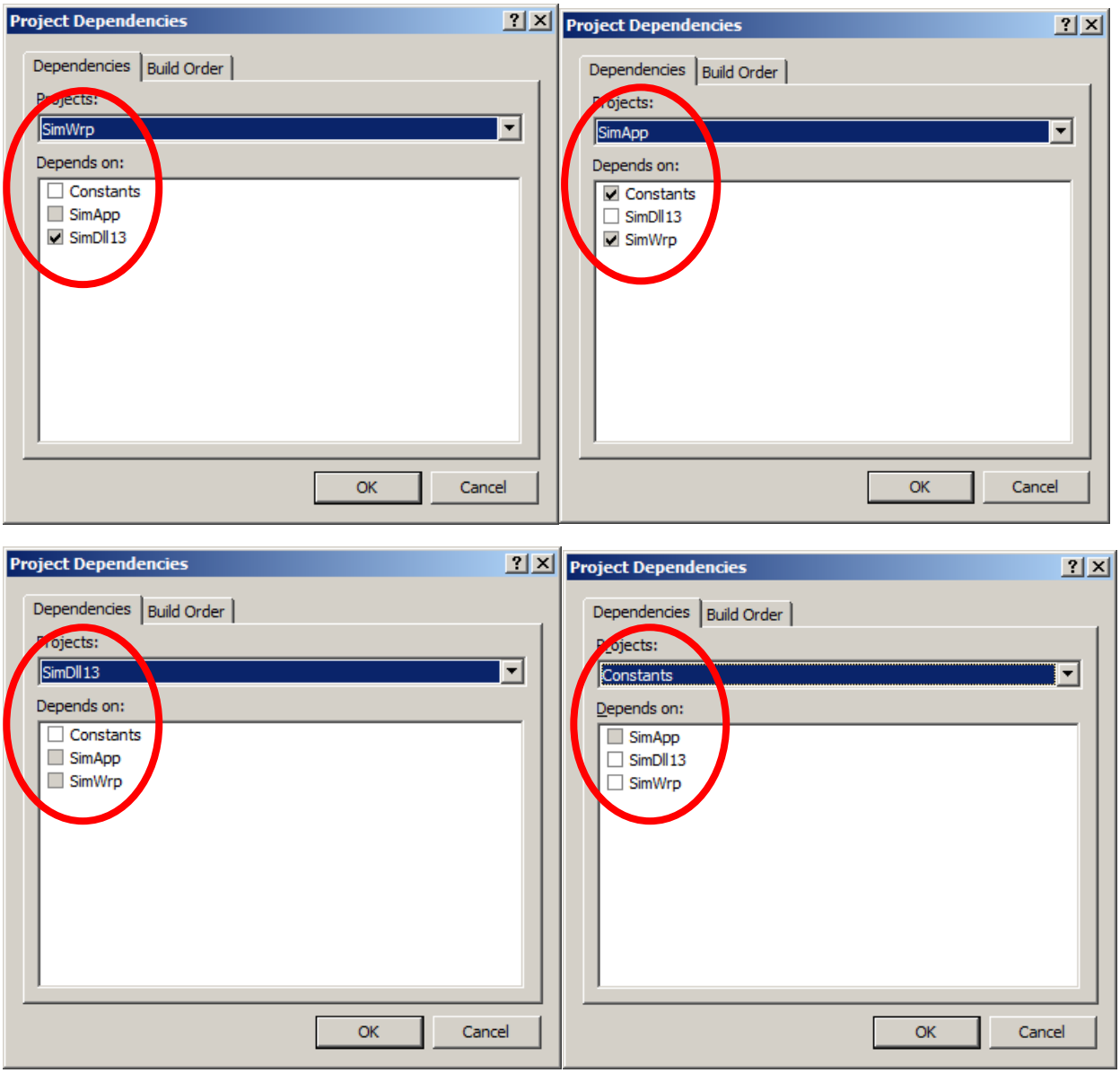


The user should take a moment to review the PROJECT settings discussed above. The first thing to be noted in the next image, is that the project SimApp is formatted bold in the Solution Explorer pane, indicating that is set as the Startup project. The SimApp project is the application and will use the other projects. This can be verified by clicking PROJECT \ Project Dependencies…, as shown below.

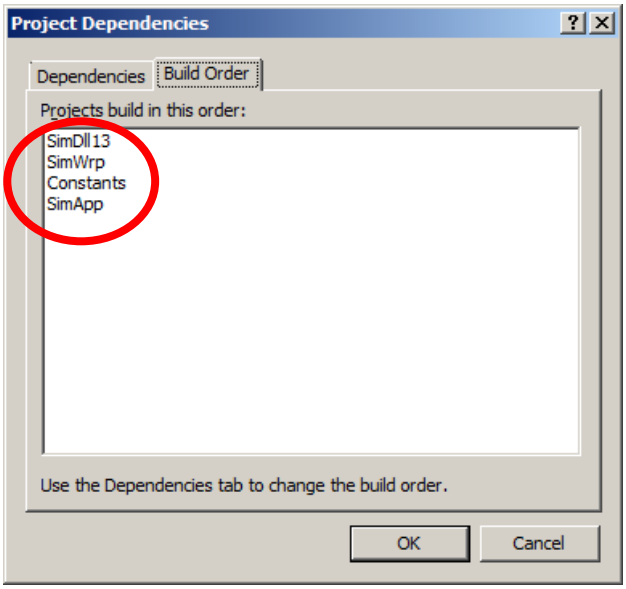


The Project Dependencies window is shown below. The next series of images show the dependencies of the various projects. The dependencies can be stated as follows:

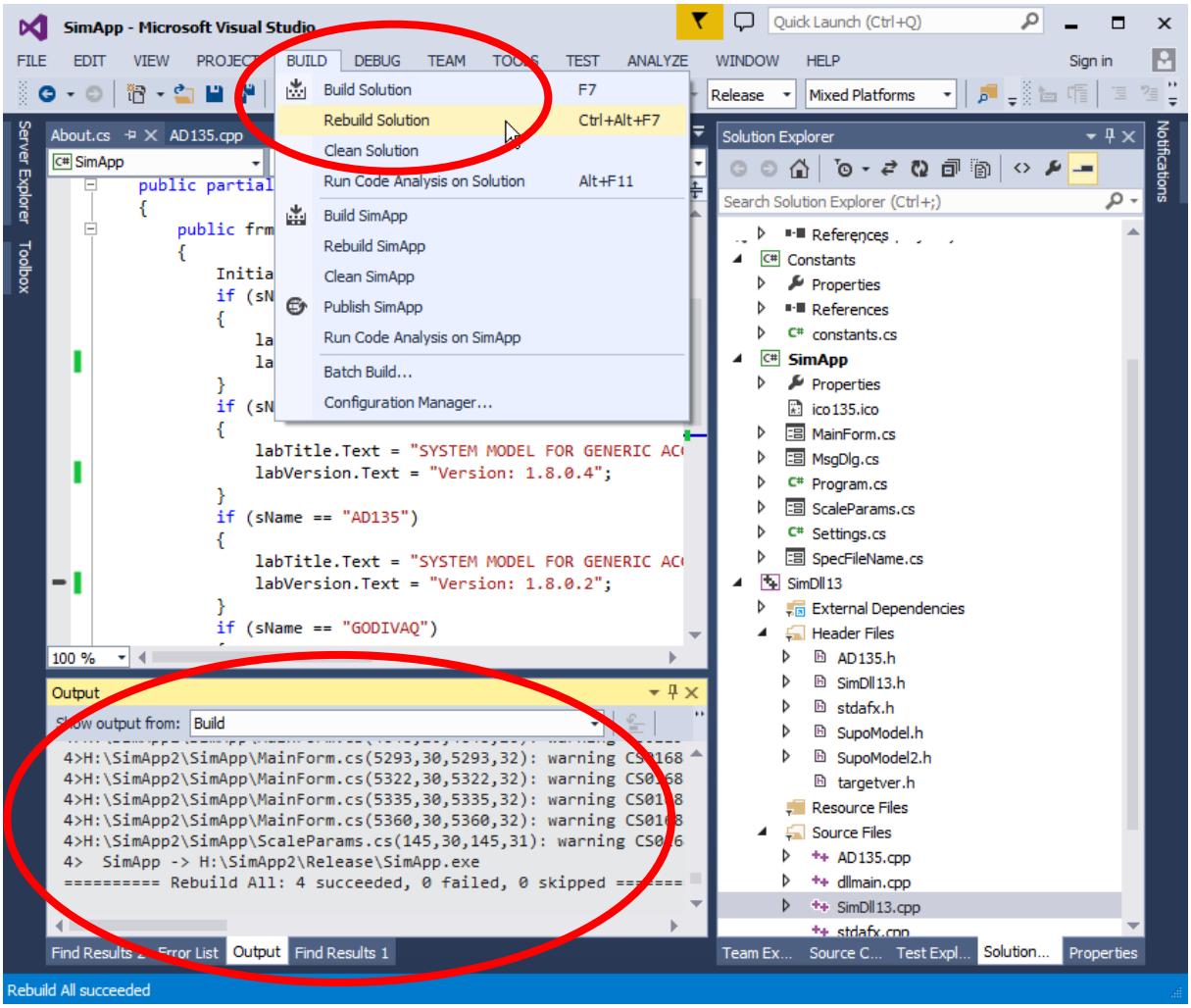
- SimWrp depends on SimDll13 – SimDll13 must compile before SimWrp
- SimApp depends on Constants and SimWrp – compile SimWrp and Constants before SimApp
- Neither SimDll13 nor Constants depend on anything – nothing needs compile before these projects



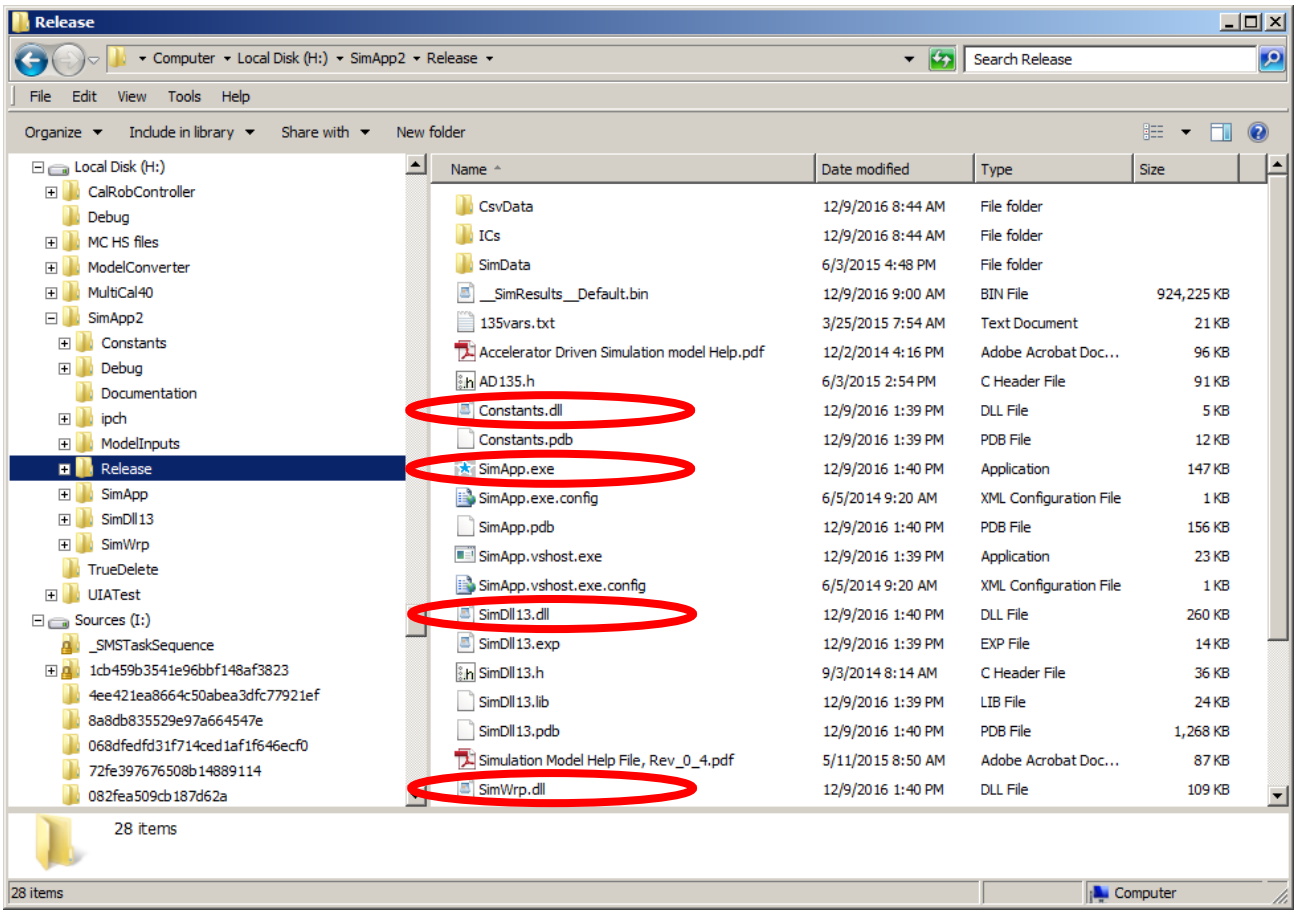
Click the "Build Order" tab in the currently open dialog. The build order of the projects reflects the dependencies we have just discussed. SimDll13 depends on nothing, but must be built before SimWrp; SimWrp and Constants must be built before SimApp; finally, SimApp must be built last as it depends either directly or indirectly on everything else. Click OK to close the dialog.



The user is ready to compile the solution SimApp. Find and click **BUILD \ Rebuild Solution**. The Output window will show any errors, or an indication of success.



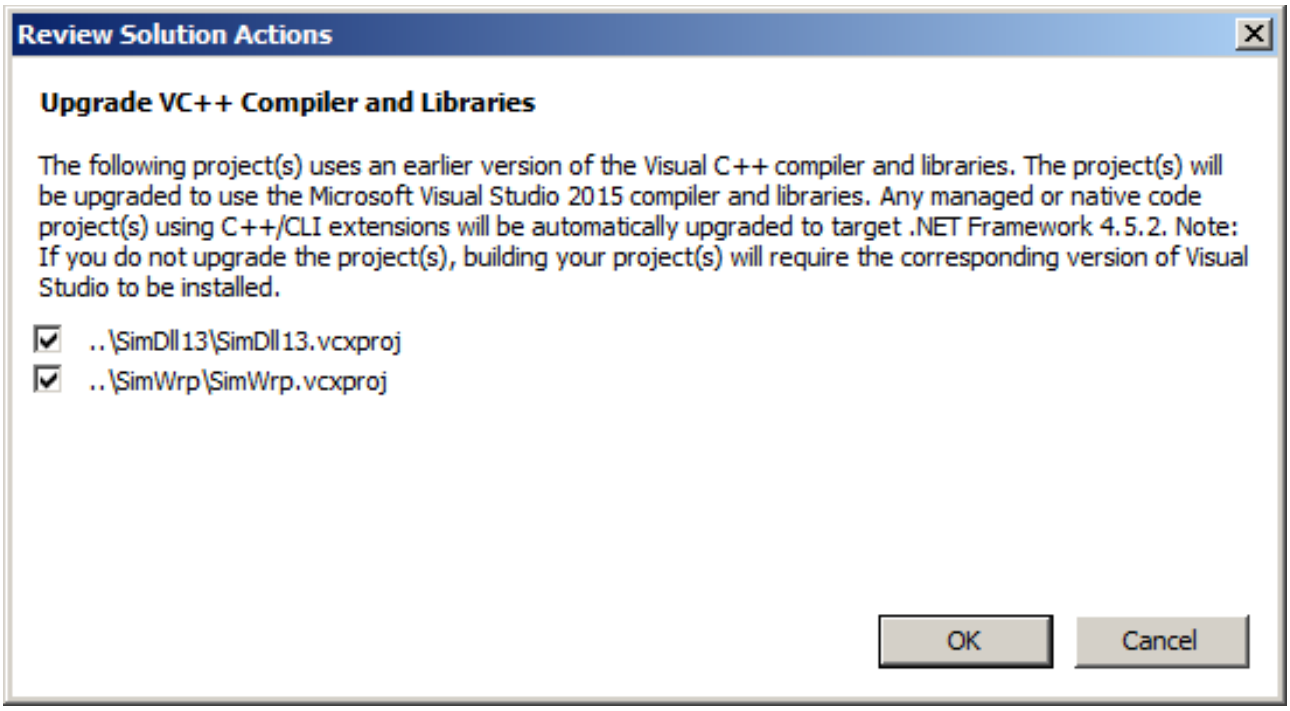
Finally, the user should verify that 4 new components are present in the Release folder, as shown below. The Date modified column indicates that the components circled in red were just created.



VS2015 Compilation Hints

Only a few special steps were required to compile SimApp using VS2015. The author's experience is summarized below.

1. The first step in converting an existing project to a new version of Visual Studio is to simply open the solution file in the new version of Visual Studio. After the messages related to source control have been answered, the following dialog will open:



Select OK. Note that .NET Framework 4.5.2 has been selected because that is the default for VS 2015. SimApp is currently using .NET Framework 4.5.1, so that is one setting we will change in the procedure outlined below.

2. In Windows Explorer, go to the folder <The Drive for your SimApp project>:\SimApp2\SimWrp. Right click on “SimWrp.vcxproj” and hover over “Open With” and select “Wordpad” from the popup menu.

3. Locate the line:

   ```
   <TargetFrameworkVersion>v4.5.2</TargetFrameworkVersion>
   ```

   And change “v4.5.2” to “v4.5.1”.

4. Save and close the file.

5. Return to the VS window opened in step 1. Focusing on this window will cause a dialog to open; click the “Reload” button. This action brings VS up-to-date with the change you made using Wordpad.

6. One other change will need to be made. Again, use Windows Explorer to locate the following folder:

   C:\Program Files (x86)\MSBuild\14.0\Bin

7. In this folder, locate the file MSBuild.exe.config. Copy this file to any other folder where we will modify it – for example, c:\temp. You may drag the file and select copy, or select the file and type Ctrl-c to copy it to the clipboard, and then, in c:\temp, click ctrl-v.

8. Once you have copied the file, right click on the original file, select "Rename" and append ".old" to the file name. Hit the return key to finish the rename operation. This step is a precaution to retain the original file, and at the same time make it possible to move the modified file back into this folder.

9. Go to C:\temp, or wherever the user has chosen to modify the copied file. Open the file using Wordpad via the right click menu.

10. Find the line:

    ```
    </runtime>
    ```

    It is near the bottom of the file. Insert the following line prior to this line:

    ```
    <enforceFIPSPolicy enabled="false"/>
    ```

11. Save and close the file.

12. Move or copy the file back to

    C:\Program Files (x86)\MSBuild\14.0\Bin

    The solution should now be ready to compile.

13. Return to VS and select **BUILD \ Rebuild All**. The solution should compile and execute successfully, but all installations of VS may be subject to unique issues, depending on the configuration of the machine in question.

Contents Detail

The relationship between the three solutions of this release are discussed in this section.

ModelConverter is used to convert model input specifications from the DESIRE modeling language to C++. See Reference 1 for more information on DESIRE. See Reference 2 for more information on the details of the model conversion process. The basic steps for using ModelConverter are discussed in a later section. ModelConverter is written entirely in C# and uses no submodule libraries.

SimApp is the Engineering Design Tool. SimApp, the Solution, consists of the following projects:

SimApp - a C# graphical user interface

SimDll13 - a C++ dynamic-link library (dll), the simulation "guts" of the application

SimWrp - a C++/CLI interface between SimApp and SimDll13

Constants – a utility C# project for specifying widely needed constants.

SimDll13 contains a class "Model". The numerical integration engine defined in SimDll13 operates on the data and methods of class Model. The output of ModelConverter is a C++ class subclassed from Model, and is produced in the typical form of a *.h and *.cpp file. This means that the class Model defines the methods and the form of the data needed to perform numerical integration, and a specific plug-in module provided by ModelConverter defines the details of a specific model within that context. One file, SimWrp.h, in the SimApp source code is modified to indicate inclusion of a subclass module. The first few lines of SimWrp.h are shown below, for inclusion of a model entitled "AD135".

```
#include "..\SimDll13\AD135.h"

using namespace System;
using namespace System::Runtime::InteropServices;

namespace SimWrp
{
        public ref class SimModel
        {
        public:
                SimModel(){ simModel = dynamic_cast<Sim::Model*> (new Sim::AD135()); }

…
}
```

Lines similar to the line starting with "#include" and the line SimModel() are the lines needed to specify any specific model to be used in SimApp, where the #include statement references the *.h file produced by ModelConverter. In addition to modifying SimWrp.h, the *.h and *.cpp files produced by ModelConverter are included in the SimDll13 project of the SimApp Solution. The *.h file produced by ModelConverter is used in one other important way. It is copied into the folders where the release and debug executables reside. It contains comments related to important variables that give those variables human readable names – the names used when manipulating data for plots and tables in the design tool. The *.h file must be present with the executable file.

The final Solution of this release is UIATest. This is a convenience tool used for regression testing. UIA stands for "User Interface Automation." UIATest runs SimApp as if a user were clicking defined sequences of operations. While this is not a general purpose tool – it was designed to run a specific set of tests useful to the author – it could be used as a template for other forms of automated testing with SimApp. The manner in which tests are automated will be discussed briefly, in case this discussion is useful to others.

To begin, users not familiar with Microsoft's User Interface (UI) Automation frame work should begin by consulting "UI Automation Overview" - https://msdn.microsoft.com/en-us/library/ms747327%28v=vs.110%29.aspx. The basic idea is that a .NET application consists of a hierarchy of components. A UI Automation tool, a distinct application from the item to be tested, attaches to a running application and can navigate the hierarchy of components to find specific elements. Thus if a test of software written with a .NET user interface consists of the following steps:

Enter value X into component A

Enter value Y into component B

Press the button C

The UI Automation framework can be used to locate the components X and Y, put values into those components, just as if a user had typed them into the interface, and then locate button C and cause it to be pressed.

- UIATest is written in C#. A handful of general methods define the test procedure:
- SetIC(array of strings defining what values to put where) – set test initial conditions
- Execute() – cause SimApp to run a simulation
- SaveSim(string defining test case ID) – save the simulation data file with a specific name based on the test ID
- ExamineFiles(string naming a root directory, string naming simulation results file) – locate results file for a given test and return path to file
- CompareResults(test case id, path to results file, pass/fail criterion) – compare values in final data record of results file to DESIRE results for the same test, stored in CSV file, return bool indicating success or failure of comparison. CompareResults also writes the outcome of each test to another CSV file, for display in Excel after the tests have been run.

Specific test cases are defined as methods, where each test case method has the following form:

Specify arbitrary number of initial conditions using successive calls to SetIC.

Call Execute.

Call SaveSim.

Call ExamineFiles.

Call CompareResults.

A method called RunTest locates the running SimApp application, opens the file to write results to, runs each test case method, and closes the output file. The end result is an application that could be run every time SimApp was changed to verify that a predefined set of regression tests still passed.