|  |
| --- |
| Level 2 |
| C# Project #2 |
| Exceptions, Arrays & Collections, Strings, and Basic Classes |

|  |
| --- |
|  |

# Contents

[Contents 1](#_Toc39741547)

[Project Overview 2](#_Toc39741548)

[Getting Started 4](#_Toc39741549)

[Adding References 4](#_Toc39741550)

[Importing Namespaces 6](#_Toc39741551)

[Implementing the Functionality 7](#_Toc39741552)

[Initialization & Configuration 7](#_Toc39741553)

[Loading Waveforms 7](#_Toc39741554)

[Building the Script 8](#_Toc39741555)

[Appendix 9](#_Toc39741556)

[Review Questions 9](#_Toc39741557)

[Project Hints 10](#_Toc39741558)

# Project Overview

In this project, you will be creating an application to generate multiple waveform files loaded from a folder on disk. The waveforms will be sequenced in the order they are loaded from disk and generated with variable amount of time in between as set by the user.

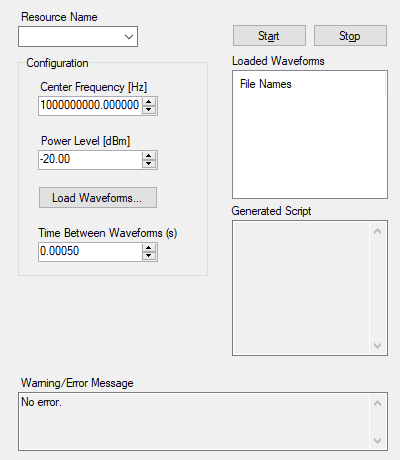


Figure 1: Application UI

The user will select a folder by pressing the **Load Waveforms…** button, launching a folder-browser dialog box. After selecting the folder, the program will find valid TDMS waveform files and display their filenames in the listbox labeled *Loaded Waveforms*. The application will silently ignore any TDMS files that do not match the format. Additionally, the program will validate that all of the TDMS files in the folder use an identical sample rate, because RFSG will require that all waveforms have an identical sample rate to be generated correctly.

If no valid TDMS files are found in the directory, the application will throw an exception that will be reported to the user. Also, if the files do not have equivalent sample rates, an exception will also be thrown and reported to the user.

When the user selects **Start**, the driver will be initialized and configured with the frequency and power level. Additionally, all valid waveform files located in the search directory will be downloaded to the generator using RFSG Playback Library. A script will be created which generates each waveform in turn, separated by a user-defined value in seconds.

An example script will be:

script myScript

repeat forever

generate waveform1

wait 500

generate waveform 2

wait 500

…

end repeat

end script

This script will be displayed in *Generated Script* and downloaded to the generator using RFSG Playback Library. Finally, RF signal generation will initiate and continue until **Stop** is pressed, which will abort generation and close the generator session.

# Getting Started

## Adding References

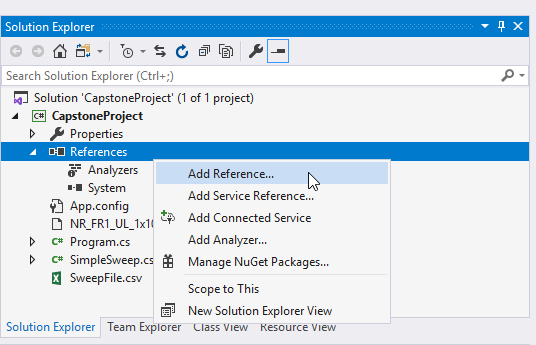
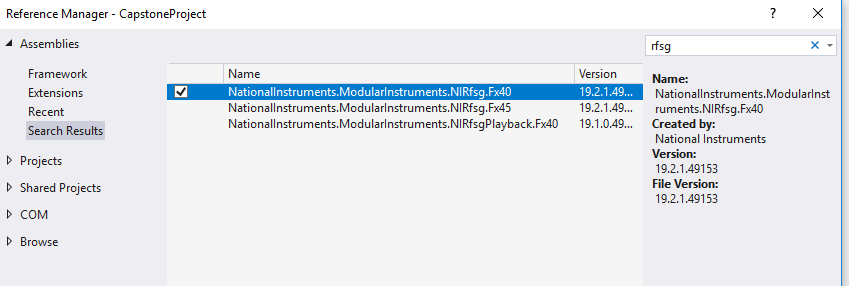
When you create a new C# project, they do not already have the necessary references to work with the NI drivers. When you develop a class that others can reuse, it is called a *class library* and is exported as an *assembly*, which has the file extension .dll. For convenience, when a class library is deployed onto a new machine, it is typically added to the [Global Assembly Cache (GAC)](https://docs.microsoft.com/en-us/dotnet/framework/app-domains/gac). This is a global registry on the computer where .NET assemblies can be registered so that other applications can use them, regardless of where they live on the computer. The closest analogy for LabVIEW is that this is similar to vi.lib or instr.lib.

You will need to add the following references to your project:



Figure 4: Necessary RF Driver Assemblies

To add an assembly to a project, follow these steps:

1. In Visual Studio in the *Solution Explorer*, right-click *References* and select **Add Reference…***Figure 5: Adding a Reference in Visual Studio 2019*
2. This will open the *Reference Manager*. On the left-hand side, select **Assemblies** to locate .NET assemblies in the GAC. *Assemblies* will show you all .NET Framework Class libraries (the *Framework* submenu) as well as assemblies added to the GAC (the *Extensions* submenu).
3. In the search bar on the right, enter the name of the assembly that you are searching for, such as **rfsg**. Once you have found the correct assembly, click the checkbox on the right to include a reference to this assembly in your project.   
     
   *Note: You may see multiple assemblies with almost the same name if you created your project to target the .NET framework v4.5 or greater. The only naming difference is that one ends in either “Fx40” or “Fx45”. This is because NI includes support for both .NET framework version 4.0 and version 4.5, and both may be installed on your system. You can select either one of them for this project.* *Figure 6: Using the Reference Manager*
4. Repeat this for the other required assembly.

## Importing Namespaces

Now that the assemblies are referenced in your project, you still need to indicate to each namespace where you will be using these assemblies that you wish to import each namespace. This keeps you from having to use the fully qualified name for each class that you wish to reference. Hence, you can transform the following code from:

NationalInstruments.ModularInstruments.NIRfsg.NIRfsg rfsg;

to the much simpler:

NIRfsg rfsg;

To do this, you need to add the using keyword along with the namespace that you wish to import. At the top of MainForm.cs, type using and enter each of the namespaces identified below:

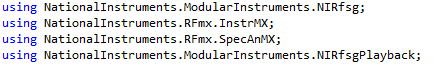
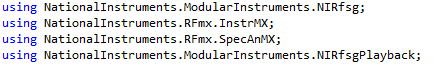


Figure 7: Namespaces to Import for the Project

Visual Studio should automatically suggest all of the namespaces above as you type them. If Visual Studio does not suggest them, you likely failed to add the required assembly to your project in the previous step.

# Implementing the Functionality

*Note: certain functionality expects for you to find the right class to use to accomplish the functionality . You are encouraged to do your own research to find the appropriate classes to use, as this will help familiarize you with available documentation. However, the focus of this exercise is the RF programming, so do not spend too much time on this. A hints section is included at the end of this document to provide help finding the appropriate classes and methods. Look for the* ***(hint)*** *tag below to know when a hint is available.*

## Initialization & Configuration

1. Update StartGeneration to create a new NIRfsg session, apply the frequency and power level, and initiate generation. Be sure to properly handle exceptions that may be thrown in the process.

#### **Initializing Drivers in C#**

In C#, instrument sessions are represented as objects. Hence, to initialize an RFmx or RFSG session, you will call the object’s constructor to create a new object representing the session. NI-RFSG devices are controlled using the NIRfsg class, and NI-RFmx devices are controlled using the RFmxInstrMX class.

1. Update StopGeneration to abort signal generation, disable output, and close the device session. Be sure to properly handle exceptions that may be thrown in the process.Test your code to validate that initializing, configuring, and stopping generation all work as expected.

## Loading Waveforms

1. Update LoadWaveforms to show a folder dialog box to prompt the user to select a folder from which to load the TDMS files. ***(hint)***
2. Using the user-selected file path from the previous step, search for all TDMS files in the specified directory. ***(hint)***
3. Add the discovered file names without extension to lsvWaveforms to display them to the user. ***(hint)*** Test your code to verify that it finds and displays TDMS files correctly. Use the *Waveform Files* directoryincluded with the project, or provide your own directory.
4. Implement the following validation logic to verify that the identified TDMS files are of the correct format and will not cause future exceptions:
   1. Since other file types can be saved with the TDMS format, your code must validate that each file found is a valid waveform file. A “valid” waveform file should be considered to be anything that is successfully loaded by the RFSG Playback Library. Invalid waveform files should be silently ignored. Test this using the *Waveform Files* directory; *BadTDMS1.tdms*should be ignored by your code. **Do this only using RFSG Playback Library.** Do not try to validate the format using raw TDMS functions (for your own sanity). ***(hint)***
   2. At least one valid TDMS waveform file must be found. If not, an exception should be created and displayed to the user. Test this by selecting a directory with no TDMS files.
   3. All valid TDMS files must share the same sample rate; otherwise, the generator will not be able to generate them all within the same script. Use NI RFSG Playback Library to read the sample rate from the file. If the files do not share the same sample rate, an exception should be created and displayed to the user. Test this by using the *Waveform Files (Unequal Sample Rates)* directory.

#### **RFSG Playback Library in C#**

Since the RFSG Playback Library is not a native part of the RFSG driver, the functionality is implemented instead in the static class NIRfsgPlayback. Like LabVIEW, methods that work with the instrument driver requires for you to pass it the RFSG session. However, the underlying C DLL cannot accept the C# object representing the RFSG session, and instead requires a pointer in memory to the actual device session.

Hence, you will perform the following actions to first get the raw instrument handle and then call the RFSG Playback library functions:



This is considered “dangerous” because now the .NET framework no longer has exclusive control over the underlying session. For example, you could close the session directly and now the object state would be invalid. “Dangerous” helps remind the developer to be cautious.

1. Update StartGeneration to read and download all valid waveforms to the signal generator before initializing generation. Test your code to verify that all waveforms are downloaded successfully. *Note – since you have not explicitly changed the generation mode yet, you should still only be generating a CW tone, which is appropriate at this point. All that you need to validate at this point is that your file loading/downloading code is correct.*

## Building the Script

1. Add a new function with a return type of string that will generate the RFSG script with all waveforms.
2. Construct a script to generate all valid found waveforms, including a wait of timeToWaitNumeric seconds between each waveform. Note that the *wait* statement in RFSG scripts expects a number in terms of samples, not seconds. Since you have validated that all sample rates are identical from the loaded files, use the sample rate from one of these files and the value from the control in seconds to determine the number of samples to wait. ***(hint)***
3. Use RFSG Playback Library to download the script to the generator. Test your program with an RFmx TXP measurement to validate that all waveforms are sequenced as expected.

# Appendix

## Review Questions

1. Now that you have learned that NI drivers are implemented as classes, describe how driver functionality you are used to in LabVIEW maps to the elements that compose classes in C#.  
     
   *[Input answer here]*
2. RFSG Playback Library doesn’t provide very good built-in functionality for validating that a TDMS waveform file is correct. Can you think of any ways to improve this by adding new functions? What could be improved about the existing functions to make this validation easier?  
     
   *[Input answer here]*
3. Why do you think the developers chose to implement the RFSG Playback Library as a static class in C#?  
     
   *[Input answer here]*
4. When is the StringBuilder useful? When would it be a poor choice to use?  
     
   *[Input answer here]*

## Project Hints

3. [FolderBrowserDialog class](https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.folderbrowserdialog?view=netframework-4.5) and [ShowDialog()](https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.commondialog.showdialog?view=netframework-4.5#System_Windows_Forms_CommonDialog_ShowDialog) method

4. [Directory.GetFiles](https://docs.microsoft.com/en-us/dotnet/api/system.io.directory.getfiles?view=netcore-3.1) method

5. [How to: Add and Remove Items with the Windows Forms ListView Control](https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-add-and-remove-items-with-the-windows-forms-listview-control) and [Path.GetFileNameWithoutExtension Method](ttps://docs.microsoft.com/en-us/dotnet/api/system.io.path.getfilenamewithoutextension?view=netcore-3.1)

6. In this section of the material, you learned about Exception handling. RFSGPlayback will throw an exception when you try to read a value from an invalid file. Try reading a property from the file and if you catch an exception, consider this file “invalid”. Make sure that your solution does not

9. See the section on the StringBuilder class on pg. 267-267 in *Murach*.