Controlling Signal Studio Using .NET API

Notice: This document contains references to Agilent. Please note that Agilent's Test and Measurement business has become Keysight Technologies. For more information, go to www.keysight.com.

KEYSIGHT TECHNOLOGIES

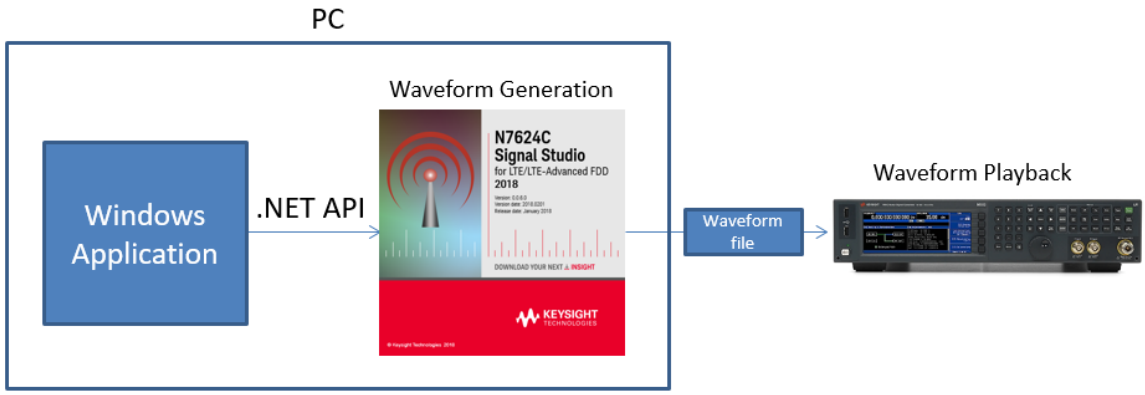
Procedure

Remote Controlling Signal Studio Using .NET API

Introduction

Signal Studio provides a .NET API (Application Programming Interface) for remote control and automation for configuring setups and generating waveforms. Using .NET API, it is possible to build a program that automates the complex tasks using Signal Studio while increasing efficiency. The system diagram for using .NET API to remotely control Signal Studio is shown in Figure 1.

Figure 1. System Diagram for Controlling Signal Studio



In this document, we will discuss creating a remote-control program using the .NET API of the Signal Studio for LTE and some practical examples. We will use Visual C# in Microsoft Visual Studio 2017 on Windows 10 (64bit) as an example in this document.

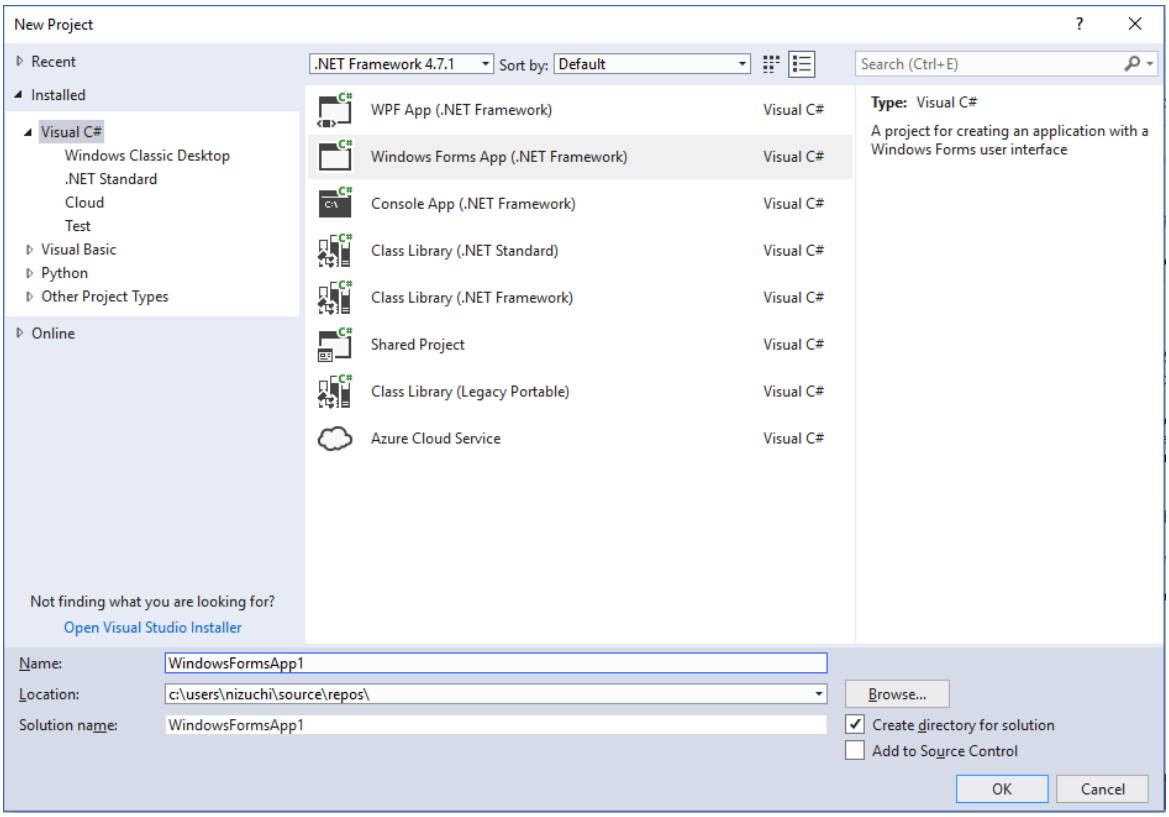
Prepare a New Project for Remote Control Program

Install the Signal Studio on your development PC, and verify that you have an MXG with a valid license installed to playback the waveforms.

First, create a new Windows Forms project to control Signal Studio using .NET API remotely (refer to Figure 2).

1. Click **File, New, Project ...**
2. In the left pane of the New Project window click **Template, Visual C#**
3. Click **.NET Framework 4.7** or **4.7.1**. If 4.7 or 4.7.1 is not listed, click **<More Frameworks..>** and download the Developer Pack.

Figure 2. New Project Window

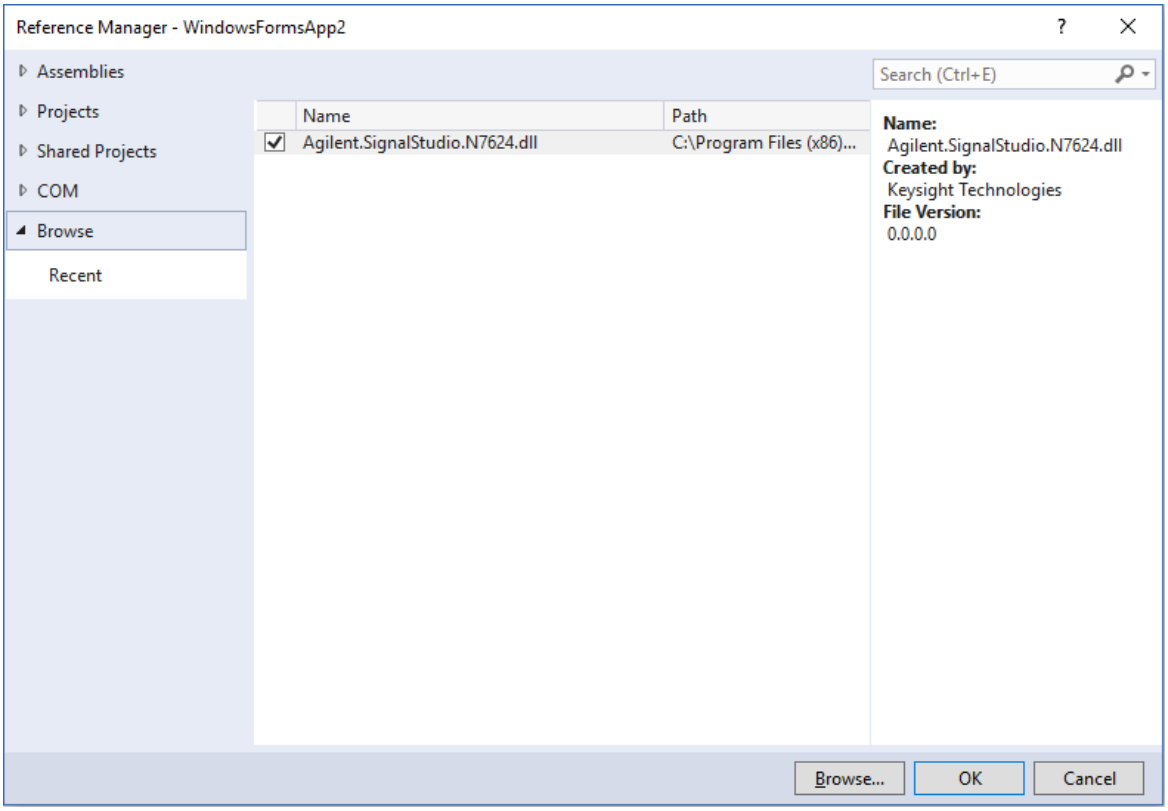


4. Click **Windows Forms App (.NET Framework)**
5. Specify project name in **Name**, then click **OK**.

Next, add DLL (Dynamic Link Library) for the Signal Studio API to the project reference.

1. To open the Reference Manager, click **Project, Add Reference...**
2. Click **Browse...**
3. For this example, select C:\Program Files (x86)\Keysight\Signal Studio\LTE-LTE-Advanced FDD 20XX.XXXX\Playback\Agilent.SignalStudio.N7624.dll
4. Click **Add**

Figure 3. Reference Manager Forms Window



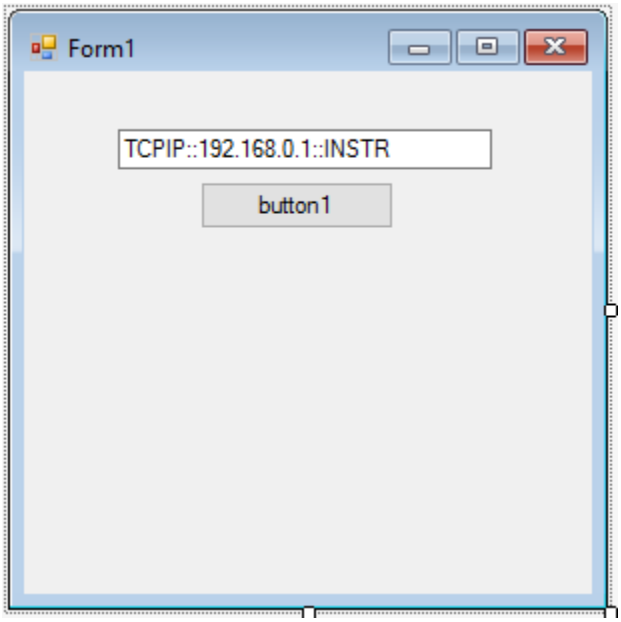
5. Click **OK** to close the Reference Manager.

NOTE Currently, the Signal Studio's API namespace retains "Agilent" name for backward compatibility reason. This will be updated to "Keysight" in the future release of Signal Studio.

Add TextBox and Button controls from the Toolbox to the Form 1 as shown in Figure 4.

1. Set the signal generator's (SG's) VISA address to the default text of the textbox.
2. Double click on **button1** to open Form1.cs source code.

Figure 4. Form1 Window



Add the following "alias" for the Signal Studio API name space in the source code.

```
using N7624 = Agilent.SignalStudio.N7624;
```

And, create a Signal Studio instance in the button1 event handler, then delete all existing carriers. The following is the code. This code will be a template to add the code samples presented in the following sections.

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }

  private N7624.N7624API mApi; //  Declare variable to store Signal Studio instance

  private void button1_Click(object sender, EventArgs e)
  {
     if (mApi == null) //  Check if Signal Studio instance is created
     {
        mApi = new N7624.N7624API(N7624.N7624API.ApplicationMode.NoGUI);
        //mApi = new N7624.N7624API(N7624.N7624API.ApplicationMode.GUI);
        //mApi.GetMainWindow().Show();
     }

     int numOfCarrier = mApi.Waveform.GetNumberOfCarriers();
     for (int i = 0; I < numOfCarrier; i++) mApi.Waveform.DeleteCarrier(i + 1);

     // Insert hardware connection code snippets here

     // Insert signal configuration code snippets here
  }
}
```

It is recommended that Signal Studio is used without the GUI, because this increases the Signal Studio software's speed. Also, some features such as digital pre-distortion (DPD) or envelope tracking (ET) may not work properly with GUI.

When you create a Signal Studio instance with GUI, the Welcome dialog box will be displayed. Once you close the dialog box by clicking **Cancel**, the program will proceed to delete existing carriers. To avoid having the Welcome dialog displaying and interrupting the current operation after Signal Studio has started:

- Click **Tools** > **Options** > click to uncheck **Display Welcome Dialog**

Configuring Signal Generator Hardware

The code sample connects to MXG at the address specified in textBox1. The address needs to be a VISA string such as “TCPIP::<ip.address>::INSTR” or “GPIB::<address>::INSTR”. If it fails to connect to the SG at the specified address, it shows a message box and will abort. Once the connection is established, it sets the following parameters.

- Frequency: 1.95GHz
- Amplitude: -10dBm
- Trigger Type: Trigger and Run Continuous
- Trigger Source: External

```
if (mApi.SetSigGenHardwareSystem(textBox1.Text) == false)
{
   MessageBox.Show("Check Connection");
   return;
}
mApi.Hardware.SignalGenerator.Frequency = 1.95e9;
mApi.Hardware.SignalGenerator.Amplitude = -10.0;
mApi.Hardware.SignalGenerator.TriggerType = N7624.API.TriggerType.Continuous;
mApi.Hardware.SignalGenerator.Continuous = N7624.API.Continuous.TriggerAndRun;
mApi.Hardware.SignalGenerator.TriggerSource = N7624.API.TriggerSource.External;
```

Configuring E-TM (Basic R9 Downlink)

The code sample configures the following conditions using Basic R9 downlink carrier, then generates and downloads the waveform to the SG.

- LTE System Bandwidth: 5MHz
- E-TM: 3.1

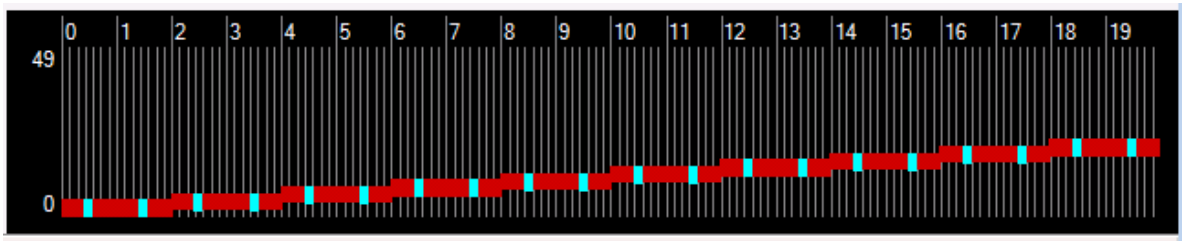
```
var R9BasicDL = (N7624.API.LTE7Basic.LTE_DL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Basic_DL);
R9BasicDL.SetTestModel(N7624.API.LTE7Basic.TMType.E_TM3_1,
   N7624.API.LTE7Basic.SystemBandwidth.F5MHz_25RB);
mApi.Download();
```

Configuring Uplink PUSCH Resource Block Allocation (Basic R9 Uplink)

The code sample sets up the Uplink Resource Block Allocation and modulation. The code sets the same parameters to all of the subframes for simplification purposes. But, you can change the RB and Modulation per subframe by modifying the contents of the loop.

- System bandwidth: 10MHz
- RB Size: 5
- RB offset: 2 RB incrementally shifted per subframe
- Modulation Type: QPSK

```
var R9BasicUL = (N7624.API.LTE7Basic.LTE_UL)
    mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Basic_UL);
R9BasicUL.SystemBandwidth = N7624.API.LTE7Basic.SystemBandwidth.F10MHz_50RB;
var puschCollection = new N7624.API.LTE7Basic.PuschResourceDef[10];
for (int i = 0; i < 10; i++)
{
    puschCollection[i] = new N7624.API.LTE7Basic.PuschResourceDef(
        0, // frame #
        i, // subframe #
        N7624.API.LTE7Basic.UL_PuschModulationType.QPSK, // Modulation Type
        Enumerable.Range(i * 2, 5).ToArray(), // 1st slot RB (Offset, Size)
        Enumerable.Range(i * 2, 5).ToArray(), // 2nd slot RB (Offset, Size)
        0, 0, // DMRS Sequence Index
        0, 0, // DMRS Group Sequence
        0, 0  // DMRS Cyclic Shift
    );
}
R9BasicUL.DeleteTransportChannel(1);
R9BasicUL.AddUL_SCH(puschCollection);
```



Configure Fixed Reference Channel (Advanced R9 Uplink)

The sample code configures FRC (Fixed Reference Channel) using “FRC Wizard”.

- LTE System bandwidth: 10MHz
- FRC: A2-1
- RB offset: 0
- Sounding RS: OFF

```
var R9AdvUL = (N7624.API.LTE7Advanced.LTE_UL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Advanced_UL);
R9AdvUL.SystemBandwidth =
   N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB;
R9AdvUL.SetFixedReferenceChannel( // Configure FRC using FRC Wizard
   N7624.API.LTE7Advanced.UlFRCType.FRC_A2_1, // FRC Type
   N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB, // Bandwidth
   0,    // RB offset
   false // SRS State
);
mApi.Download();
```

Configure Transmission Configuration for UL_SCH (Advanced R9 Uplink)

The sample code configures UL-SCH’s transmission configuration per subframe.

- System bandwidth: 10MHz
- RB size: 10
- RB offset: 10
- MCS Index: 1

```
var R9AdvUL = (N7624.API.LTE7Advanced.LTE_UL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Advanced_UL);
R9AdvUL.SystemBandwidth =
   N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB;
for (int i = 0; i < 10; i++)
{
   R9AdvUL.SCH.TxParam(i).RBOffset = 10;
   R9AdvUL.SCH.TxParam(i).RBSize = 10;
   R9AdvUL.SCH.TxParam(i).MCSIndex = N7624.API.LTE7Advanced.UlschMCSIndexType.MCS_1;
}
mApi.Download();
```

Configure Transmission Configuration for PUCCH (Advanced R9 Uplink)

- System bandwidth: 10MHz
- PUCCH format: 2
- n(1)PUCCH: 1

```
var R9AdvUL = (N7624.API.LTE7Advanced.LTE_UL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Advanced_UL);
R9AdvUL.SystemBandwidth = N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB;
R9AdvUL.SCH.StateEnabled = false;
R9AdvUL.UCI.StateEnabled = true;
for (int i = 0; i < 10; i++)
{
   R9AdvUL.UCI.TxParam(i).Format = N7624.API.LTE7Advanced.PucchFormat.Format_2;
   R9AdvUL.UCI.TxParam(i).n1PUCCH = 1;
}
mApi.Download();
```

Configure PRACH Preambles (Basic R9 PRACH)

The sample code configures PRACH preambles using “PRACH Wizard”. The following parameters are set.

- System bandwidth: 10MHz
- RB offset: 0
- PRACH Config Indx: 14
- Cyclic Shift Set: Unrestricted
- Preamble ID: 32
- Logical Root Sequence Index: 22
- Ncs Index: 1

```
var R9PRACH = (N7624.API.LTE7Basic.LTE_PRACH)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Basic_PRACH);
R9PRACH.PRACHConfiguration(
   N7624.API.LTE7Basic.SystemBandwidth.F10MHz_50RB, // System bandwidth
   0, // RB Offset
   14, // PRACH Config Index
   N7624.API.LTE7Basic.CyclicShiftSet.Unrestricted, // Cyclic Shift Set (High Speed flag)
   32, // Preamble Index
   N7624.API.LTE7Basic.AutoConfiguration.PRACHTestPreambles, // Auto-Configuration
   22, // Logical Root Sequence Index
   1, // Ncs Configuration Index
   true // Increase timing offset
);
mApi.Download();
```

PDSCH Tx Configuration (Advanced R9 Downlink)

This sample code configures per subframe transmission configuration for PDSCH. It sets the following parameters for each subframes:

- RB Size: 25
- RB Offset: 10
- MCS Index: 1

```
var R9AdvDL = (N7624.API.LTE7Advanced.LTE_DL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Advanced_DL);
R9AdvDL.SystemBandwidth = N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB;
for (int i = 0; i < R9AdvDL.SCH(3).TxParamLength; i++)
{
   R9AdvDL.SCH(3).TxParam(i).State = true;
   R9AdvDL.SCH(3).TxParam(i).RBSize = 25;
   R9AdvDL.SCH(3).TxParam(i).RBOffset = 10;
   R9AdvDL.SCH(3).TxParam(i).MCSIndex1 =
      N7624.API.LTE7Advanced.DlschMCSIndexType.MCS_1;
}
mApi.Download();
```

DCI Format 0 (UL Scheduling) Tx Configuration (Advanced R9 Downlink)

The sample code configures DCI format 0 to schedule UE. The following parameters are configured per subframe.

- RB Size: 25
- RB Offset: 10
- MCS Index: 10
- PUSCH TPC: #1

DCI TxParam takes 3 parameters:

- 1st parameter specifies DCI number. The DCI number used for UL scheduling is DCI#9.
- 2nd parameter specifies the frame number and
- 3rd specifies the subframe number. The sample code converts Tx Configuration Length (in subframes) to frames and includes the subframe number with division and modulo operation.

```
var R9AdvDL = (N7624.API.LTE7Advanced.LTE_DL)
    mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Advanced_DL);
R9AdvDL.SystemBandwidth = N7624.API.LTE7Advanced.SystemBandwidth.F10MHz_50RB;
for (int i = 0; i < R9AdvDL.SCH(3).TxParamLength; i++)
{
    R9AdvDL.DCI.TxParam(9, i / 10, i % 10).State = true;
    R9AdvDL.DCI.TxParam(9, i / 10, i % 10).RBSize = 25;
    R9AdvDL.DCI.TxParam(9, i / 10, i % 10).RBOffset = 10;
    R9AdvDL.DCI.TxParam(9, i / 10, i % 10).MCS =
        N7624.API.LTE7Advanced.PdcchMCS.MCS_10;
    R9AdvDL.DCI.TxParam(9, i / 10, i % 10).TPCPusch =
        N7624.API.LTE7Advanced.TPCPusch.TPC_1;
}
mApi.Download();
```

Configuring LTE-A carrier aggregation (Advanced R10 Downlink)

The sample code configures 2CC Downlink Carrier Aggregation with the following parameters.

- CC1: 10MHz, 64QAM, -4.95MHz offset
- CC2: 10MHz, 16QAM, +4.95MHz offset

The code configures 2CC carrier aggregation with both component carriers with 64QAM modulation. Then, it takes reference to CC#2 and change modulation type to 16QAM. Using same technique, you can configure other parameters for both CC#1 and #2.

```
var R10AdvDL = (N7624.API.LTEAAdvanced.LTEA_DL_AGG)
    mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTEAAdvanced_AggregationDL);
R10AdvDL.CarrierConfiguration =
    N7624.API.LTEAAdvanced.DownlinkCarrierConfigration.CC2_10MHz_10MHz_64QAM;
var CC2 = (N7624.API.LTEAAdvanced.LTEA_DL_COMPONENT)R10AdvDL.GetCarrierParameter(2);
CC2.ChannelConfiguration =
    N7624.API.LTEAAdvanced.DownlinkConfiguration.FullFilledQPSK10M;
mApi.Download();
```

Using Advanced LTE-A Pro Carrier for NB-IoT

To use Advantest LTE-A Pro carrier, you need to add the following “Using” alias at the top of the code. (LTE-A Pro carrier uses “Keysight” name space.)

```
using K_N7624 = Keysight.SignalStudio.N7624;
```

The following code configures stand-alone NB-IoT uplink carrier.

```
var LTEAProUL_CA =
(K_N7624.Api.LteAProAdvanced.LteAProUplinkAggregation)mApi.Waveform.AddCarrier(Agilent.SignalStudio
.N7624.API.CarrierType.LteAProFddAdvancedUplinkNbiot);
var CC1 =
(K_N7624.Api.LteAProAdvanced.LteAProUplinkNbiotComponentCarrier)LTEAProUL_CA.GetCarrierParameter(1)
;

CC1.OperationMode = K_N7624.Api.LteAProAdvanced.OperationModeUL.StandAlone;
CC1.NarrowbandCellId = 0;
CC1.Sch.NpuschFormat = K_N7624.Api.LteAProAdvanced.NpuschFormat.Format1;
CC1.Sch.SubcarrierSpacing = K_N7624.Api.LteAProAdvanced.SubcarrierSpacing.F15kHz;
CC1.Sch.NumberOfSubcarriersInResourceUnit =
K_N7624.Api.LteAProAdvanced.NumberOfSubcarriersInResourceUnit.Sc1;
CC1.Sch.SubcarrierOffsetInResourceBlock = 0;
CC1.Sch.NumberOfResourceUnits = K_N7624.Api.LteAProAdvanced.NumberOfResourceUnits.N2;
CC1.Sch.NumberOfRepetition = K_N7624.Api.LteAProAdvanced.NpuschNumberOfRepetition.Repetition1;
CC1.Sch.NpuschStartingSubframe = 2;
CC1.Sch.McsIndex = 0;
mApi.Download();
```

MSR example (Library Waveform)

The sample code configures LTE and W-CDMA combined waveform. To do this, you need to generate W-CDMA waveform using N7600B Signal Studio for W-CDMA/HSPA+. This sample code assumes the W-CDMA waveform (TM1 w/ 64DPCH) is imported to Signal Studio for LTE using the Library Manager.

To access the Library Manager:

- Click **Tools** > **Waveform Library menu**

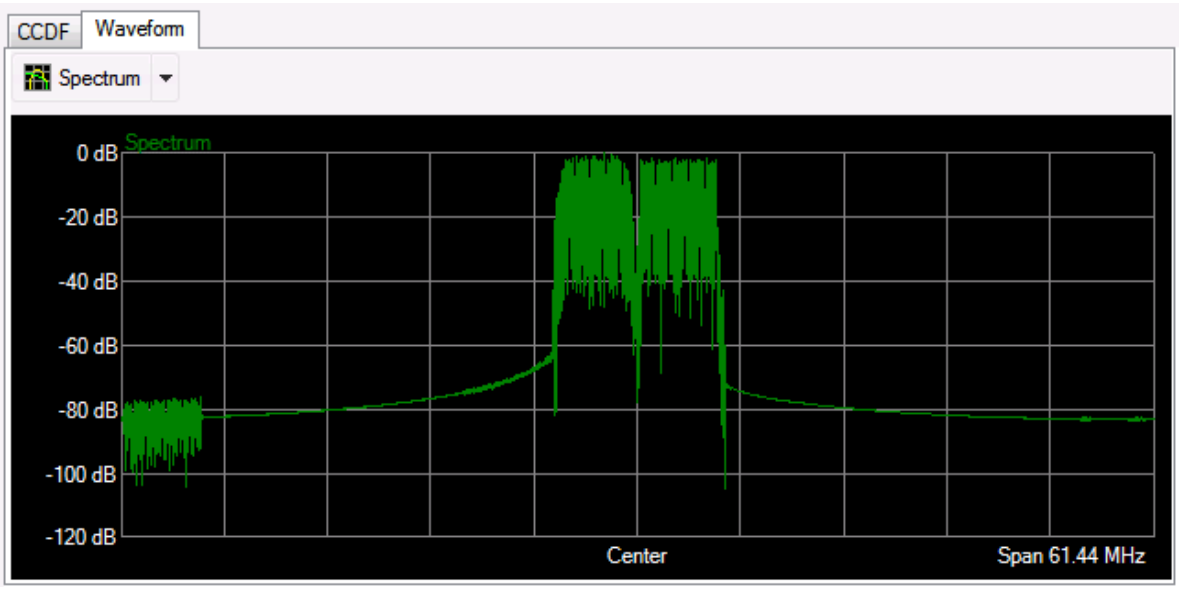
You can download the Library waveform package from Keysight.com that contains W-CDMA, TD-SCDMA, and GSM/EDGE sample waveforms for use in Library Waveform feature.

www.keysight.com/find/msrwaveform

You also need to manually copy "`importer.dll`" to the project.

1. Click **Project**
2. Add **Existing Item...**
3. From Signal Studio's folder under Program Files (x86) add **importer.dll**
4. Change **Copy to output directory** to **Copy if newer**

```
var UTRA = (N7624.API.LibraryWaveform.LibraryWaveformPlayback)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LibraryWaveform);
UTRA.SourceWaveformFileName = @"WaveformLibrary\MSR\W-CDMA\Test Model 1 + 64 DPCH";
UTRA.FrequencyOffset = -2.5e6;
var EUTRA = (N7624.API.LTE7Basic.LTE_DL)
   mApi.Waveform.AddCarrier(N7624.API.CarrierType.LTE7Basic_DL);
EUTRA.SetTestModel(N7624.API.LTE7Basic.TMType.E_TM3_1,
   N7624.API.LTE7Basic.SystemBandwidth.F5MHz_25RB);
UTRA.FrequencyOffset = +2.5e6;
mApi.Download();
```



Additional Resources

For further reference, you can refer to Signal Studio's API Reference from the Help menu. Also, there are various kinds of sample programs that can be built and executed in the following folders if you installed the Signal Studio for LTE on your PC.

N7624C	C:\Users\<user>\AppData\Roaming\Keysight\Signal Studio\LTE-LTE-Advanced FDD 20XX.XXXX\Samples\
N7625C	C:\Users\<user>\AppData\Roaming\Keysight\Signal Studio\LTE-LTE-Advanced TDD 20XX.XXXX\Samples\

Edition 1, February 2018

www.keysight.com