# Keysight Technologies Signal Generators

E4438C RF

E8267D PSG Microwave

N5162A/82A MXG RF

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

Creating and Downloading Waveform Files

# Notices

## Trademark Acknowledgments

## Manual Part Number

## Print Date

## Warranty

## Technology Licenses

## Restricted Rights Legend

## Safety Notices

### CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

### WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

## Table of Contents

Contents

# Creating and Downloading Waveform Files

| | |
|---|---|
| NOTE | The ability to play externally created waveform data in the signal generator is available only in the N5162A/82A with Option 651, 652 or 654, E4438C ESG Vector Signal Generators with Option 001, 002, 601, or 602, and E8267D PSG Vector Signal Generators with Option 601 or 602. |

On the Keysight MXG, the internal baseband generator speed upgrade Options 670, 671, and 672 are option upgrades that **require** Option 651 and 652 to have been loaded at the factory (refer to the **Data Sheet** for more information). Any references to 651, 652, or 654 are inclusive of 671, 672, and 674.

For the N5162A, the softkey menus and features mentioned in this manual are only available through the Web–Enabled MXG or through SCPI commands. Refer to **Programming Guide** and to the **SCPI Command Reference**.

This manual explains how to create Arb–based waveform data and download it into the signal generator. This information is also available in the signal generator's Programming Guide.

**KEYSIGHT**
TECHNOLOGIES

## Overview of Downloading and Extracting Waveform Files

The signal generator lets you download and extract waveform files. You can create these files either external to the signal generator or by using one of the internal modulation formats (ESG/PSG only). The signal generator also accepts waveforms files created for the earlier E443xB ESG signal generator models. For file extractions, the signal generator encrypts the waveform file information. The exception to encrypted file extraction is user–created I/Q data. The signal generator lets you extract this type of file unencrypted. After extracting a waveform file, you can download it into another Keysight signal generator that has the same option or software license required to play it. Waveform files consist of three items:

1. I/Q data
2. Marker data
3. File header

NOTE    This order of download is required, as the I/Q data downloads results in the overwriting of all of these three parts of the file.

The signal generator automatically creates the marker file and the file header if the two items are *not* part of the download. In this situation, the signal generator sets the file header information to unspecified (no settings saved) and sets all markers to zero (off).

There are three ways to download waveform files: FTP, programmatically or using one of three available free download utilities created by Keysight Technologies:

— N7622A Signal Studio Toolkit 2
   *http://www.keysight.com/find/signalstudio*

— Keysight Waveform Download Assistant for use only with MATLAB
   *http://www.keysight.com/find/downloadassistant*

— Intuilink for Keysight PSG/ESG Signal Generators
   *http://www.keysight.com/find/intuilink*

NOTE    Intuilink is **not** available for the Keysight MXG.

## Waveform Data Requirements

To be successful in downloading files, you must first create the data in the required format.

— Signed 2's complement

— 2–byte integer values

— Input data range of -32768 to 32767

— Minimum of 60 samples per waveform (60 I and 60 Q data points)

— Interleaved I and Q data

— Big endian byte order

— The same name for the marker, header, and I/Q file

This is only a requirement if you create and download a marker file and or file header, otherwise the signal generator automatically creates the marker file and or file header using the I/Q data file name.

| NOTE | FTP can be used without programming commands to transfer files from the PC to the signal generator or from the signal generator to the PC. |
| --- | --- |

For more information, see "Waveform Structure" on page 13.

For more information on waveform data, see "Understanding Waveform Data" on page 5.

# Understanding Waveform Data

The signal generator accepts binary data formatted into a binary I/Q file. This section explains the necessary components of the binary data, which uses ones and zeros to represent a value.

## Bits and Bytes

Binary data uses the base–two number system. The location of each bit within the data represents a value that uses base two raised to a power ($2^{n-1}$). The exponent is n - 1 because the first position is zero. The first bit position, zero, is located at the far right. To find the decimal value of the binary data, sum the value of each location:

$$1101 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$
$$= (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$$
$$= 13 \text{ (decimal value)}$$

Notice that the exponent identifies the bit position within the data, and we read the data from right to left.

The signal generator accepts data in the form of bytes. Bytes are groups of eight bits:

$$01101110 = (0 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1)$$
$$+ (0 \times 2^0) = 110 \text{ (decimal value)}$$

The maximum value for a single unsigned byte is 255 (11111111 or $2^8$-1), but you can use multiple bytes to represent larger values. The following shows two bytes and the resulting integer value:

$$01101110\ 10110011 = 28339 \text{ (decimal value)}$$

The maximum value for two unsigned bytes is 65535. Since binary strings lengthen as the value increases, it is common to show binary values using hexadecimal (hex) values (base 16), which are shorter. The value 65535 in hex is FFFF. Hexadecimal consists of the values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. In decimal, hex values range from 0 to 15 (F). It takes 4 bits to represent a single hex value.

| | | | | |
|---|---|---|---|---|
| 1 = 0001 | 2 = 0010 | 3 = 0011 | 4 = 0100 | 5 = 0101 |
| 6 = 0110 | 7 = 0111 | 8 = 1000 | 9 = 1001 | A = 1010 |
| B = 1011 | C = 1100 | D = 1101 | E = 1110 | F = 1111 |

For I and Q data, the signal generator uses two bytes to represent an integer value.

## LSB and MSB (Bit Order)

Within groups (strings) of bits, we designate the order of the bits by identifying which bit has the highest value and which has the lowest value by its location in the bit string. The following is an example of this order.

| | |
|---|---|
| Most Significant Bit (MSB) | This bit has the highest value (greatest weight) and is located at the far left of the bit string. |
| Least Significant Bit (LSB) | This bit has the lowest value (bit position zero) and is located at the far right of the bit string. |

Bit Position   15 14 13 12 11 10   9   8     7   6   5   4   3 2 1   0

Data   1   0   1   1   0   1   1   1    1   1   1   0   1   0   0   1

MSB                  LSB
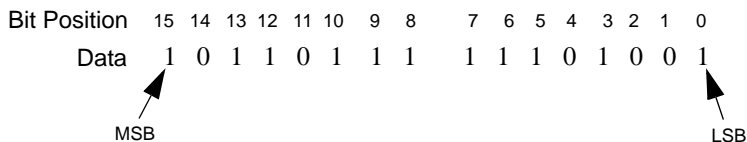
Because we are using 2 bytes of data, the LSB appears in the second byte.

## Little Endian and Big Endian (Byte Order)

When you use multiple bytes (as required for the waveform data), you must identify their order. This is similar to identifying the order of bits by LSB and MSB. To identify byte order, use the terms little endian and big endian. These terms are used by designers of computer processors.

**Little Endian Order**

The lowest order byte that contains bits 0–7 comes first.

Bit Position   7   6   5   4   3   2   1   0    15 14 13 12 11 10   9   8

Data   1   1   1   0   1   0   0   1    1   0   1   1   0   1   1   1     Hex values = E9 B7

LSB       MSB
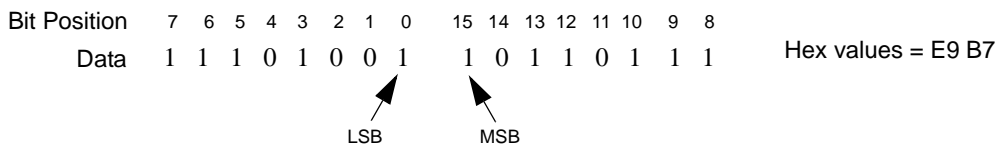
**Big Endian Order**

The highest order byte that contains bits 8–15 comes first.

Bit Position   15 14 13 12 11 10   9   8    7   6   5   4   3 2 1   0

Data   1   0   1   1   0   1   1   1    1   1   1   0   1   0   0   1     Hex values = B7 E9

MSB                  LSB
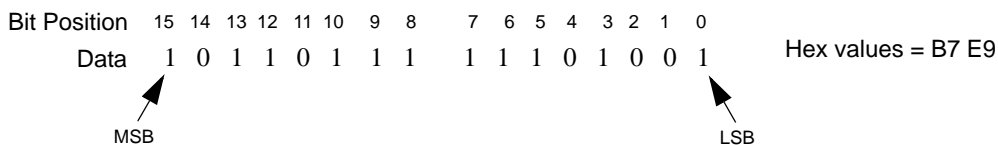
Notice in the previous figure that the LSB and MSB positioning changes with the byte order. In little endian order, the LSB and MSB are next to each other in the bit sequence.

**NOTE**
For I/Q data downloads, the signal generator requires big endian order. For each I/Q data point, the signal generator uses four bytes (two integer values), two bytes for the I point and two bytes for the Q point.

The byte order, little endian or big endian, depends on the type of processor used with your development platform. Intel processors and its clones use little endian. Sun and Motorola processors use big endian. The Apple PowerPC processor, while big endian oriented, also supports the little endian order. Always refer to the processor's manufacturer to determine the order they use for bytes and if they support both, to understand how to ensure that you are using the correct byte order.

Development platforms include any product that creates and saves waveform data to a file. This includes Keysight Technologies Advanced Design System EDA software, C++, MATLAB, and so forth.

The byte order describes how the system processor stores integer values as binary data in memory. If you output data from a little endian system to a text file (ASCII text), the values are the same as viewed from a big endian system. The order only becomes important when you use the data in binary format, as is done when downloading data to the signal generator.

## Byte Swapping

While the processor for the development platform determines the byte order, the recipient of the data may require the bytes in the reverse order. In this situation, you must reverse the byte order before downloading the data. This is commonly referred to as byte swapping. You can swap bytes either programmatically or by using either the Keysight Technologies Intuilink for ESG/PSG Signal Generator software, or the Signal Studio Toolkit 2 software. For the signal generator, byte swapping is the method to change the byte order of little endian to big endian. For more information on little endian and big endian order, see "Little Endian and Big Endian (Byte Order)" on page 6.

The following figure shows the concept of byte swapping for the signal generator. Remember that we can represent data in hex format (4 bits per hex value), so each byte (8 bits) in the figure shows two example hex values.



|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Little Endian | E9 | B7 | 53 | 2A |

16–bit integer values (2 bytes = 1 integer value)

I data = bytes 0 and 1
Q data = bytes 2 and 3

|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Big Endian | B7 | E9 | 2A | 53 |

I    Q
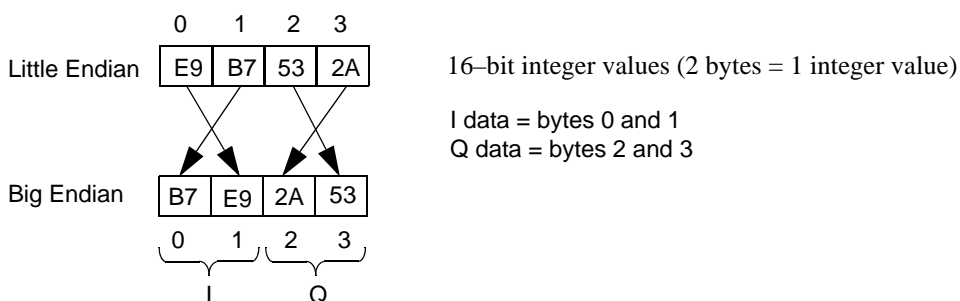
To correctly swap bytes, you must group the data to maintain the I and Q values. One common method is to break the two–byte integer into one–byte character values (0–255). Character values use 8 bits (1 byte) to identify a character. Remember that the maximum unsigned 8–bit value is 255 ($2^8$ - 1). Changing the data into character codes groups the data into bytes. The next step is then to swap the bytes to align with big endian order.

**NOTE**    The signal generator always assumes that downloaded data is in big endian order, so there is no data order check. Downloading data in little endian order will produce an undesired output signal.

## DAC Input Values

The signal generator uses a 16–bit DAC (digital–to–analog convertor) to process each of the 2–byte integer values for the I and Q data points. The DAC determines the range of input values required from the I/Q data. Remember that with 16 bits we have a range of 0–65535, but the signal generator divides this range between positive and negative values:

— 32767 = positive full scale output
— 0 = 0 volts
— -32768 = negative full scale output

Because the DAC's range uses both positive and negative values, the signal generator requires signed input values. The following list illustrates the DAC's input value range.

| Voltage | DAC Range | Input Range | Binary Data | Hex Data |
|---------|-----------|-------------|-------------|----------|
| Vmax | 65535 | 32767 | 01111111 11111111 | 7FFF |
| | 32768 | 1 | 00000000 00000001 | 0001 |
| 0 Volts | 32767 | 0 | 00000000 00000000 | 0000 |
| | 32766 | -1 | 11111111 11111111 | FFFF |
| Vmin | 0 | −32768 | 10000000 00000000 | 8000 |

Notice that it takes only 15 bits ($2^{15}$) to reach the Vmax (positive) or Vmin (negative) values. The MSB determines the sign of the value. This is covered in .

## Using E443xB ESG DAC Input Values

In this section, the words *signal generator* with or without a model number refer to an N5162A/82A Keysight MXG, E4438C ESG, E8267D PSG. The signal generator input values differ from those of the earlier E443xB ESG models. For the E443xB models, the input values are all positive (unsigned) and the data is contained within 14 bits plus 2 bits for markers. This means that the E443xB DAC has a smaller range:

— 0 = negative full scale output
— 8192 = 0 volts
— 16383 = positive full scale output

Although the signal generator uses signed input values, it accepts unsigned data created for the E443xB and converts it to the proper DAC values. To download an E443xB files to the signal generator, use the same command syntax as for the E443xB models. For more information on downloading E443xB files, see "Downloading E443xB Signal Generator Files" on page 54.

## Scaling DAC Values

The signal generator uses an interpolation algorithm (sampling between the I/Q data points) when reconstructing the waveform. For common waveforms, this interpolation can cause overshoot, which may exceed the limits of the signal process path's internal number representation, causing arithmatic overload. This will be reported as either a data path overload error (N5162A/82A) or a DAC over–range error condition (E4438C/E8267D). Because of the interpolation, the error condition can occur even when all the I and Q values are within the DAC input range. To avoid the DAC over–range problem, you must scale (reduce) the I and Q input values, so that any overshoot remains within the DAC range.

Whenever you interchange files between signal generator models, ensure
that all scaling is adequate for that signal generator's waveform.
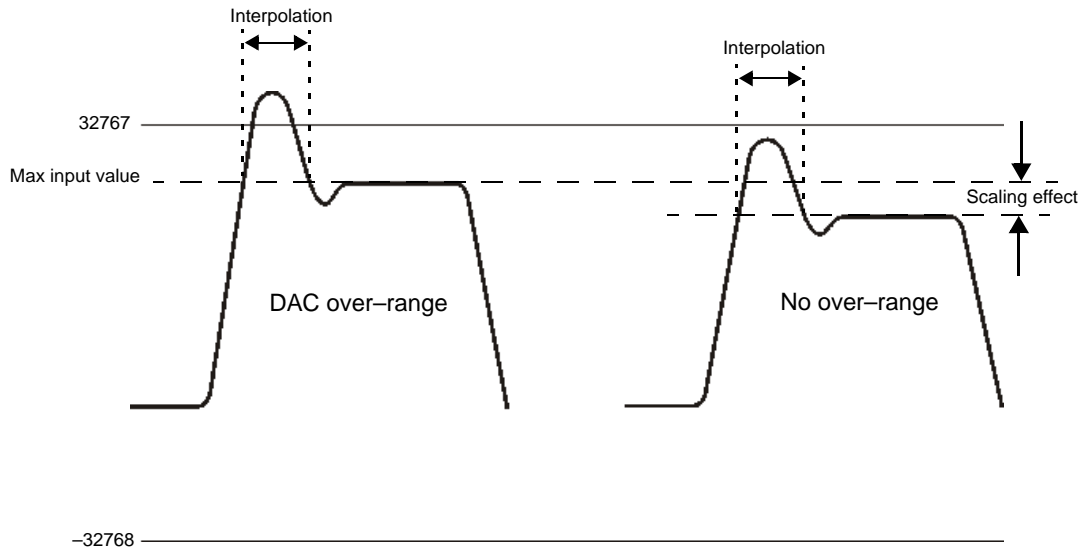


There is no single scaling value that is optimal for all waveforms. To achieve
the maximum dynamic range, select the largest scaling value that does not
result in a DAC over–range error. There are two ways to scale the I/Q data:

— Reduce the input values for the DAC.
— Use the SCPI command `:RADio:ARB:RSCaling <val>` to set the
waveform amplitude as a percentage of full scale.

**NOTE** The signal generator factory preset for scaling is 70%. If you reduce the DAC
input values, ensure that you set the signal generator scaling
(:RADio:ARB:RSCaling) to an appropriate setting that accounts for the
reduced values.

To further minimize overshoot problems, use the correct FIR filter for your
signal type and adjust your sample rate to accommodate the filter response.

**NOTE** FIR filter capability is only available on the N5162A/82A with Option 651,
652, or 654, the E4438C with Option 001, 002, 601, or 602, and on the
E8267D with Option 601 or 602.

## 2's Complement Data Format

The signal generator requires signed values for the input data. For binary data, two's complement is a way to represent positive and negative values. The most significant bit (MSB) determines the sign.

— 0 equals a positive value (01011011 = 91 decimal)

— 1 equals a negative value (10100101 = -91 decimal)

Like decimal values, if you sum the binary positive and negative values, you get zero. The one difference with binary values is that you have a carry, which is ignored. The following shows how to calculate the two's complement using 16–bits. The process is the same for both positive and negative values.

Convert the decimal value to binary.

```
23710 = 01011100 10011110
```

Notice that 15 bits (0–14) determine the value and bit 16 (MS2B) indicates a positive value.
Invert the bits (1 becomes 0 and 0 becomes 1).

```
10100011 01100001
```

Add one to the inverted bits. Adding one makes it a two's complement of the original binary value.

```
  10100011 01100001
+ 00000000 00000001
  10100011 01100010
```

The MSB of the resultant is one, indicating a negative value (-23710).
Test the results by summing the binary positive and negative values; when correct, they produce zero.

```
  01011100 10011110
+ 10100011 01100010
  00000000 00000000
```

## I and Q Interleaving

When you create the waveform data, the I and Q data points typically reside in separate arrays or files. The signal generator requires a single I/Q file for waveform data playback. The process of interleaving creates a single array with alternating I and Q data points, with the Q data following the I data. This array is then downloaded to the signal generator as a binary file. The interleaved file comprises the waveform data points where each set of data points, one I data point and one Q data point, represents one I/Q waveform point.
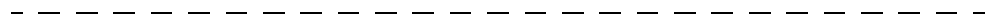
**NOTE**     The signal generator can accept separate I and Q files created for the earlier E443xB ESG models. For more information on downloading E443xB files, see .

The following figure illustrates interleaving I and Q data. Remember that it takes two bytes (16 bits) to represent one I or Q data point.

|  | MSB | LSB | MSB | LSB |
|---|---|---|---|---|

I Data    Binary    11001010    01110110    01110111    00111110

             Hex    CA          76          77          3E

Q Data    Binary    11101001    11001010    01011110    01110010

             Hex    E9          CA          5E          72

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**Interleaved Binary Data**

Waveform data point                 Waveform data point

11001010 01110110    11101001 11001010    01110111 00111110    01011110 01110010

     I Data            Q Data            I Data            Q Data
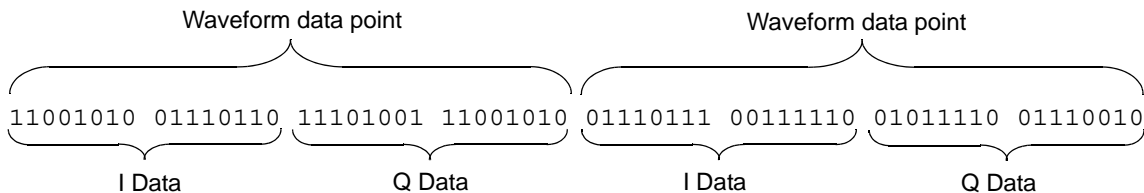
**Interleaved Hex Data**

Waveform    Waveform
data point    data point

CA   76   E9   CA   77   3E   5E   72

I Data   Q Data   I Data   Q Data
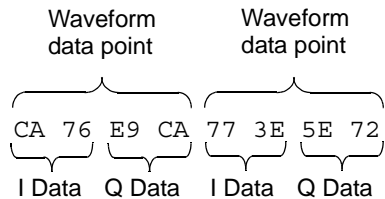
# Waveform Structure

To play back waveforms, the signal generator uses data from the following three files:

— File header
— Marker file
— I/Q file

All three files have the same name, the name of the I/Q data file, but the signal generator stores each file in its respective directory (headers, markers, and waveform). For information on file extractions, see "Commands for Downloading and Extracting Waveform Data" on page 26.

## File Header

The file header contains settings for the ARB modulation format such as sample rate, marker polarity, I/Q modulation attenuator setting and so forth. When you create and download I/Q data, the signal generator automatically creates a file header with all saved parameters set to unspecified. With unspecified header settings, the waveform either uses the signal generator default settings, or if a waveform was previously played, the settings from that waveform. Ensure that you configure and save the file header settings for each waveform.

NOTE      If you have no RF output when you play back a waveform, ensure that the marker RF blanking function has not been set for any of the markers. The marker RF blanking function is a header parameter that can be inadvertently set active for a marker by a previous waveform. To check for and turn RF blanking off manually, refer to "Configuring the Pulse/RF Blank (Keysight MXG)" on page 113 and "Configuring the Pulse/RF Blank (ESG/PSG)" on page 113.

## Marker File

The marker file uses one byte per I/Q waveform point to set the state of the four markers either on (1) or off (0) for each I/Q point. When a marker is active (on), it provides an output trigger signal to the rear panel EVENT 1 connector (Marker 1 only) or and the AUX IO, event 2 connector pin (Markers 1, 2, 3, or 4), that corresponds to the active marker number. (For more information on active markers and their output trigger signal location, refer to your signal generator's **User's Guide**.) Because markers are set at each waveform point, the marker file contains the same number of bytes as there are waveform points. For example, for 200 waveform points, the marker file contains 200 bytes.

Although a marker point is one byte, the signal generator uses only bits 0–3 to configure the markers; bits 4–7 are reserved and set to zero. The following example shows a marker byte.

```
                           4  3  2  1  Marker Number Position
Marker Byte      0000 1 0 1 1
```

Reserved

– – – – – – – – –
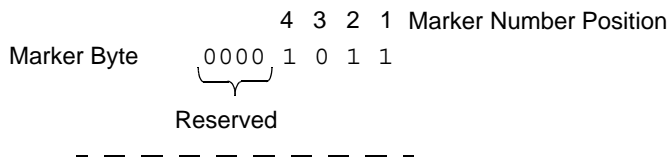
**Example of Setting a Marker Byte**

```
Binary  0000 0101
 Hex   05
```

Sets markers 1 and 3 on for a waveform point

The following example shows a marker binary file (all values in hex) for a waveform with 200 points. Notice the first marker point, 0f, shows all four markers on for only the first waveform point.

```
00000000: 0f 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    0f = All markers on
00000010: 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    01 = Marker 1 on
00000020: 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
00000030: 01 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05    05 = Markers 1 and 3 on
00000040: 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05    04 = Marker 3 on
00000050: 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05
00000060: 05 05 05 05 04 04 04 04 04 04 04 04 04 04 04 04    00 = No active markers
00000070: 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04
00000080: 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04
00000090: 04 04 04 04 04 04 00 00 00 00 00 00 00 00 00 00
000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000c0: 00 00 00 00 00 00 00 00 ▮
```

If you create your own marker file, its name must be the same as the waveform file. If you download I/Q data without a marker file, the signal generator automatically creates a marker file with all points set to zero. For more information on markers, see the *User's Guide*.

NOTE          Downloading marker data using a file name that currently resides on the signal generator overwrites the existing marker file without affecting the I/Q (waveform) file. However, downloading just the I/Q data with the same file name as an existing I/Q file also overwrites the existing marker file setting all bits to zero.

## I/Q File

The I/Q file contains the interleaved I and Q data points (signed 16–bit integers for each I and Q data point). Each I/Q point equals one waveform point. The signal generator stores the I/Q data in the waveform directory.

**NOTE**      If you download I/Q data using a file name that currently resides on the signal generator, it also overwrites the existing marker file setting all bits to zero and the file header setting all parameters to unspecified.

## Waveform

A waveform consists of samples. When you select a waveform for playback, the signal generator loads settings from the file header. When the ARB is on, it creates the waveform samples from the data in the marker and I/Q (waveform) files. The file header, while required, does not affect the number of bytes that compose a waveform sample. One sample contains five bytes:

| **I/Q Data** | **+** | **Marker Data** | **=** | **1 Waveform Sample** |
|---|---|---|---|---|
| 2 bytes I   2 bytes Q (16 bits)   (16 bits) | | 1byte (8 bits) Bits 4–7 reserved—Bits 0–3 set | | 5 bytes |

To create a waveform, the signal generator requires a minimum of 60 samples. To help minimize signal imperfections, use an even number of samples (for information on waveform continuity, see "Waveform Phase Continuity" on page 16). When you store waveforms, the signal generator saves changes to the waveform file, marker file, and file header.
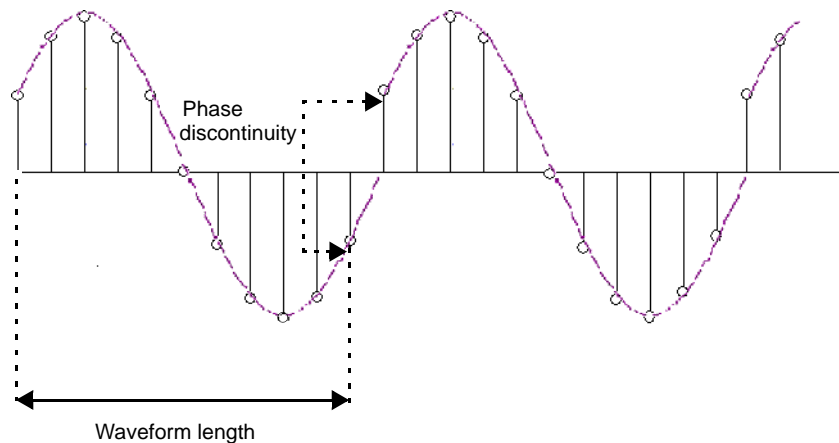
# Waveform Phase Continuity

## Phase Discontinuity, Distortion, and Spectral Regrowth

The most common arbitrary waveform generation use case is to play back a waveform that is finite in length and repeat it continuously. Although often overlooked, a phase discontinuity between the end of a waveform and the beginning of the next repetition can lead to periodic spectral regrowth and distortion.

For example, the sampled sinewave segment in the following figure may have been simulated in software or captured off the air and sampled. It is an accurate sinewave for the time period it occupies, however the waveform does not occupy an entire period of the sinewave or some multiple thereof. Therefore, when repeatedly playing back the waveform by an arbitrary waveform generator, a phase discontinuity is introduced at the transition point between the beginning and the end of the waveform.

Repetitions with abrupt phase changes result in high frequency spectral regrowth. In the case of playing back the sinewave samples, the phase discontinuity produces a noticeable increase in distortion components in addition to the line spectra normally representative of a single sinewave.

**Sampled Sinewave with Phase Discontinuity**



## Avoiding Phase Discontinuities

You can easily avoid phase discontinuities for periodic waveforms by simulating an integer number of cycles when you create your waveform segment.

NOTE            If there are N samples in a complete cycle, only the first N–1 samples are stored in the waveform segment. Therefore, when continuously playing back the segment, the first and Nth waveform samples are always the same, preserving the periodicity of the waveform.

By adding off time at the beginning of the waveform and subtracting an equivalent amount of off time from the end of the waveform, you can address phase discontinuity for TDMA or pulsed periodic waveforms. Consequently, when the waveform repeats, the lack of signal present avoids the issue of phase discontinuity.

However, if the period of the waveform exceeds the waveform playback memory available in the arbitrary waveform generator, a periodic phase discontinuity could be unavoidable. N5110B Baseband Studio for Waveform Capture and Playback alleviates this concern because it does not rely on the signal generator waveform memory. It streams data either from the PC hard drive or the installed PCI card for N5110B enabling very large data streams. This eliminates any restrictions associated with waveform memory to correct for repetitive phase discontinuities. Only the memory capacity of the hard drive or the PCI card limits the waveform size.

**Sampled Sinewave with No Discontinuity**
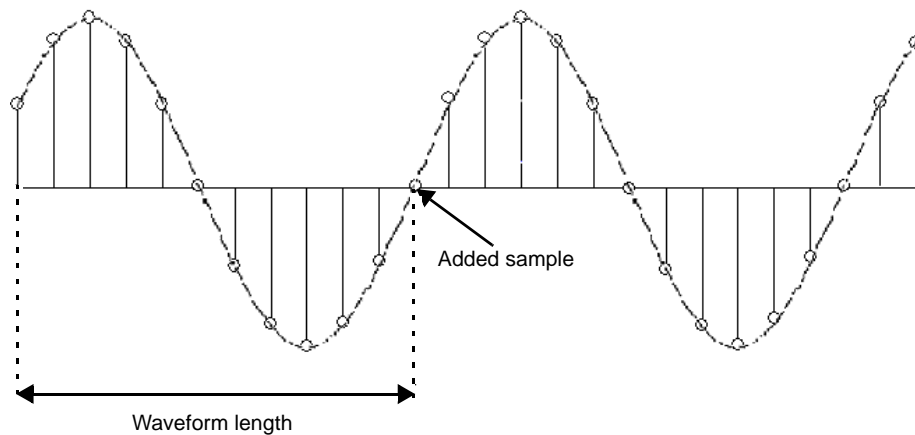


The following figures illustrate the influence a single sample can have. The generated 3–tone test signal requires 100 samples in the waveform to maintain periodicity for all three tones. The measurement on the left shows the effect of using the first 99 samples rather than all 100 samples. Notice all the distortion products (at levels up to -35 dBc) introduced in addition to the wanted 3–tone
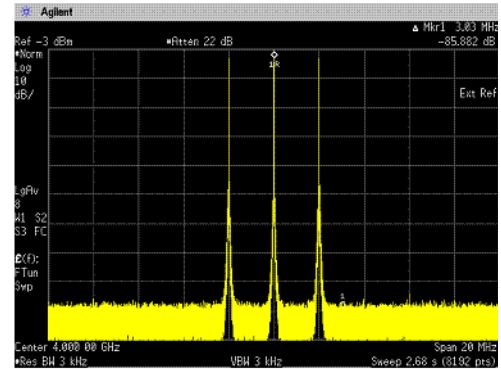
signal. The measurement on the right shows the same waveform using all 100 samples to maintain periodicity and avoid a phase discontinuity. Maintaining periodicity removes the distortion products.

**Phase Discontinuity**
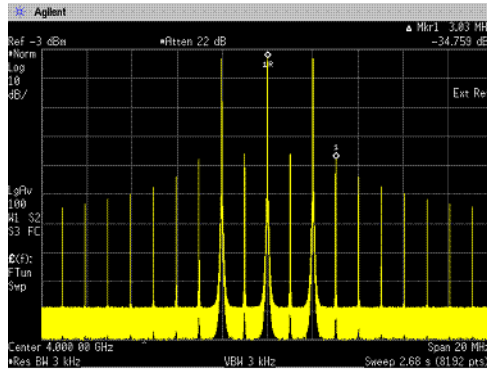
**Phase Continuity**



3–tone – 20 MHz Bandwidth
Measured distortion = 35 dBc

3–tone – 20 MHz Bandwidth
Measured distortion = 86 dBc

# Waveform Memory

The signal generator provides two types of memory, volatile and non–volatile. You can download files to either memory type.

NOTE    The MXG's ARB Waveform File Cache is limited to 128 files. Consequently, once the 128 file cache limit has been reached, the waveform switching speed will be much slower for files loaded into the volatile waveform memory (BBG).

Volatile    Random access memory that does not survive cycling of the signal generator power. This memory is commonly referred to as waveform memory (WFM1) or waveform playback memory. To play back waveforms, they must reside in volatile memory. The following file types share this memory:

Table 1    Signal Generators and Volatile Memory Types

| Volatile Memory Type | Model of Signal Generator | | |
|---|---|---|---|
| | N5162A, N5182A with Option 651, 652, or 654 | E4438C with Option 001[1], 002[1], 601, or 602 | E8267D Option 601 or 602 |
| I/Q | x | x | x |
| Marker | x | x | x |
| File header | x | x | x |
| User PRAM | – | x | x |

1. Options 001 and 002 apply only to the E4438C ESG.

Non–volatile    Storage memory where files survive cycling the signal generator power. Files remain until overwritten or deleted. To play back waveforms after cycling the signal generator power, you must load waveforms from non–volatile waveform memory (NVWFM) to volatile waveform memory (WFM1). On the Keysight MXG the

non–volatile memory is referred to as internal media and external media. The following file types share this memory:

Table 2                    Signal Generators and Non–Volatile Memory Types

| Non–Volatile Memory Type | Model of Signal Generator | | |
|---|---|---|---|
| | N5162A, N5182A with Option 651, 652, or 654 | E4438C with Option 001, 002, 601, or 602 | E8267D Option 601 or 602 |
| I/Q | x | x | x |
| Marker | x | x | x |
| File header | x | x | x |
| Sweep List | x | x | x |
| User Data | x | x | x |
| User PRAM | – | x | x |
| Instrument State | x | x | x |
| Waveform Sequences (multiple I/Q files played together) | x | x | x |

The following figure on shows the locations within the signal generator for volatile and non–volatile waveform data.

Figure 1
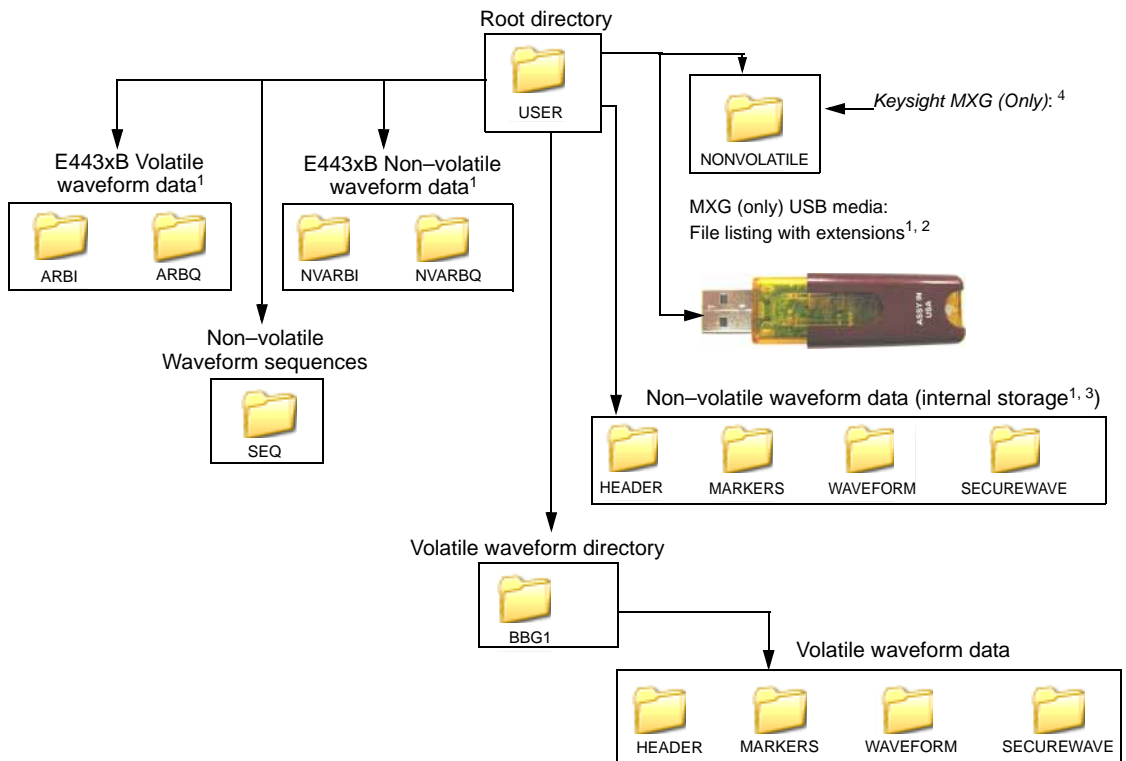


1For information on using the E443xB directories, see "Downloading E443xB Signal Generator Files" on page 54.
2The Keysight MXG uses an optional "USB media" to store non–volatile waveform data.
3The Keysight MXG internal non–volatile memory is referred to as "internal storage".
4This NONVOLATILE directory shows the files with the same extensions as the USB media and is useful with ftp.

## Memory Allocation

### Volatile Memory

The signal generator allocates volatile memory in blocks of 1024 bytes. For example, a waveform file with 60 samples (the minimum number of samples) has 300 bytes (5 bytes per sample x 60 samples), but the signal generator allocates 1024 bytes of memory. If a waveform is too large to fit into 1024 bytes, the signal generator allocates additional memory in multiples of 1024 bytes. For example, the signal generator allocates 3072 bytes of memory for a waveform with 500 samples (2500 bytes).

3 x 1024 bytes = 3072 bytes of memory

As shown in the examples, waveforms can cause the signal generator to allocate more memory than what is actually used, which decreases the amount of available memory.

| NOTE | In the first block of data of volatile memory that is allocated for each waveform file, the file header requires 512 bytes (N5162A/82A) or 256 bytes (E4438C/E8267D). |

## Non–Volatile Memory (Keysight MXG)

| NOTE | If the Keysight MXG's external USB flash memory port is used, the USB flash memory can provide actual physical storage of non–volatile data in the SECUREWAVE directory versus the "virtual" only data.<br><br>ARB waveform encryption of proprietary information is supported on the external non–volatile USB flash memory.<br><br>To copy **unencrypted** data files from an external media (as in USB Flash Drive [UFD]) for playing on a signal generator, the full filename extension is required (i.e., .MARKER, .HEADER, .WAVEFORM, etc.). For more information on unencrypted data, refer to "Commands for Downloading and Extracting Waveform Data" on page 26. For more information on how to work with files, refer to the **User's Guide**.<br><br>To copy **compatible licensed** encrypted data files (i.e., `.SECUREWAVE`) from an external media, download (copy) the files to the signal generator (refer to the **User's Guide** for information on how to work with files). When using the external media along with the signal generator's **Use as** or **Copy File to Instrument** softkey menus, encrypted data files can be automatically detected by the Keysight MXG, regardless of the suffix (e.g. .wfm, .wvfm, and **no suffix**, etc.). These various waveform files can be selected and played by the Keysight MXG. For more information on encrypted data, refer to "Commands for Downloading and Extracting Waveform Data" on page 26. When using the **Copy File to Instrument**, the signal generator prompts the user to select between BBG Memory and Internal Storage memories as locations to copy the files. |

On the N5162A/82A, non–volatile files are stored on the non–volatile internal signal generator memory (internal storage) or to an USB media, if available.

The Keysight MXG non–volatile internal memory is allocated according to a Microsoft compatible file allocation table (FAT) file system. The Keysight MXG signal generator allocates non–volatile memory in clusters according to the drive size (see Table 3 on page 23). For example, referring to Table 3 on page 23, if the drive size is 15 MB and if the file is less than or equal to 4K bytes, the file uses only one 4 KB cluster of memory. For files larger than 4 KB,

and with a drive size of 15 MB, the signal generator allocates additional memory in multiples of 4KB clusters. For example, a file that has 21,538 bytes consumes 6 memory clusters (24,000 bytes).

For more information on default cluster sizes for FAT file structures, refer to Table 3 on page 23 and to **http://support.microsoft.com/**.

Table 3                                Drive Size (logical volume)

| Drive Size (logical volume) | Cluster Size (Bytes) (Minimum Allocation Size) |
| --- | --- |
| 0 MB – 15 MB | 4K |
| 16 MB – 127 MB | 2K |
| 128 MB – 255 MB | 4K |
| 256 MB – 511 MB | 8K |
| 512 MB – 1023 MB | 16K |
| 1024 MB – 2048 MB | 32K |
| 2048 MB – 4096 MB | 64K |
| 4096 MB – 8192 MB | 128K |
| 8192 MB – 16384 MB | 256K |

### Non–Volatile Memory (ESG/PSG)

The ESG/PSG signal generators allocate non–volatile memory in blocks of 512 bytes. For files less than or equal to 512 bytes, the file uses only one block of memory. For files larger than 512 bytes, the signal generator allocates additional memory in multiples of 512 byte blocks. For example, a file that has 21,538 bytes consumes 43 memory blocks (22,016 bytes).

## Memory Size

The amount of available memory, volatile and non–volatile, varies by option and the size of the other files that share the memory. When we refer to waveform files, we state the memory size in samples (one sample equals five bytes). The ESG and PSG baseband generator (BBG) options (001, 002, 601, or 602) and the Keysight MXG baseband generator (BBG) Option (651, 652, and 654) contain the waveform playback memory. Refer to Tables 4 on page 24 through Table 6 on page 25 for the maximum available memory.

## Volatile and Non–Volatile Memory (N5162A/82A)

Table 4                     N5162A/82A Volatile (BBG) and Non–Volatile (Internal Storage and USB Media) Memory

| Volatile (BBG) Memory | | Non–Volatile (Internal Storage and USB Media) Memory | |
|---|---|---|---|
| Option | Size | Option | Size |
| **N5162A/82A**[1] | | | |
| **651/652/654 (BBG)** | 8 MSa (40 MB) | **Standard (N5182A)** | 800 MSa (4 GB)[2] |
| **019 (BBG)** | 64 MSa (320 MB) | **USB Flash Drive (UFD)** | **user determined** |

1. On the N5162A/82A, 512 bytes is reserved for each waveform's header file (i.e. The largest waveform that could be played with a N5162A/82A with Option 019 (320 MB) is: 320 MB – 512 bytes = 319,999,488 MB.)
2. For serial numbers <MY4818xxxx, US4818xxxx, and SG4818xxxx, the persistent memory value = 512 MB.

## Volatile Memory and Non–Volatile Memory (E4438C and E8267D Only)

**NOTE**          When considering volatile memory, it is not necessary to keep track of marker data, as this memory is consumed automatically and proportionally to the I/Q data created (i.e. 1 marker byte for every 4 bytes of I/Q data).

On the E4438C and E8267D, the fixed file system overhead on the signal generator is used to store directory information. When calculating the available volatile memory for waveform files it is important to consider the fixed file system overhead for the volatile memory of your signal generator.

Table 5                     Fixed File System Overhead

| Volatile (WFM1) Memory and Fixed File Overhead | | | | |
|---|---|---|---|---|
| Option | Size | Maximum Number of Files (MaxNumFiles) | Memory (Bytes) Used for Fixed File System Overhead[1] [16 + (44 x MaxNumFiles)] | Memory Available for Waveform Samples |
| **E4438C and E8267D** | | | | |
| **001, 601 (BBG)** | 8 MSa (40 MB) | 1024 | 46,080 | **8,377,088 Samples** |
| **002 (BBG)** | 32 MSa (160 MB) | 4096 | 181,248 | **33,509,120 Samples** |

Table 5                    Fixed File System Overhead

| Volatile (WFM1) Memory and Fixed File Overhead | | | | |
|---|---|---|---|---|
| **602 (BBG)** | 64 MSa (320 MB) | 8192 | 361,472 | **67,018,496 Samples** |

1. The expression [16 + (44 x MaxNumFiles)] has been rounded up to nearest memory block (1024 bytes). (To find the I/Q waveform sample size, this resulting value needs to be divided by 4.)

Table 6                    E4438C and E8267D Non–Volatile (NVWFM) Memory

| Non–Volatile (NVWFM) Memory | |
|---|---|
| **Option** | **Size** |
| **E4438C and E8267D** | |
| **Standard** | 3 MSa (15 MB) |
| **005 (Hard disk)** | 1.2 GSa (6 GB) |

# Commands for Downloading and Extracting Waveform Data

You can download I/Q data, the associated file header, and marker file information (collectively called waveform data) into volatile or non–volatile memory. For information on waveform structure, see "Waveform Structure" on page 13.

The signal generator provides the option of downloading waveform data either for extraction or not for extraction. When you extract waveform data, the signal generator may require it to be read out in encrypted form. The SCPI download commands determine whether the waveform data is extractable.

If you use SCPI commands to download waveform data to be extracted later, you must use the `MEM:DATA:UNPRotected` command. If you use FTP commands, no special command syntax is necessary.

| NOTE | On the N5162A/82A, `:MEM:DATA` enables file extraction. On the N5162A/82A the `:MEM:DATA:UNPRotected` command is **not** required to enable file extraction. For more information, refer to the **SCPI Command Reference**. |
|------|------|

You can download or extract waveform data created in any of the following ways:

— with signal simulation software, such as MATLAB or Advanced Design System (ADS)
— with advanced programming languages, such as C++, VB or VEE
— with Keysight Signal Studio software
— with the signal generator

## Waveform Data Encryption

You can download encrypted waveform data extracted from one signal generator into another signal generator with the same option or software license for the modulation format. You can also extract encrypted waveform data created with software such as MATLAB or ADS, providing the data was downloaded to the signal generator using the proper command.

When you generate a waveform from the signal generator's internal ARB modulation format (ESG/PSG only), the resulting waveform data is automatically stored in volatile memory and is available for extraction as an encrypted file.

When you download an exported waveform using a Keysight Signal Studio software product, you can use the FTP process and the securewave directory or SCPI commands, to extract the encrypted file to the non–volatile memory on the signal generator. Refer to "File Transfer Methods" on page 28.

### Encrypted I/Q Files and the Securewave Directory

The signal generator uses the `securewave` directory to perform file encryption (extraction) and decryption (downloads). The `securewave` directory is not an actual storage directory, but rather a portal for the encryption and decryption process. While the `securewave` directory contains file names, these are actually pointers to the true files located in signal generator memory (volatile or non–volatile). When you download an encrypted file, the `securewave` directory decrypts the file and unpackages the contents into its file header, I/Q data, and marker data. When you extract a file, the `securewave` directory packages the file header, I/Q data, and marker data and encrypts the waveform data file. When you extract the waveform file (I/Q data file), it includes the other two files, so there is no need to extract each one individually.

The signal generator uses the following `securewave` directory paths for file extractions and encrypted file downloads:

| | |
|---|---|
| Volatile | */user/bbg1/securewave/file_name* or *swfm:file_name* |
| Non–volatile | */user/securewave* or *snvwfm1:file_name* |

| | |
|---|---|
| **NOTE** | To extract files (other than user–created I/Q files) and to download encrypted files, you *must* use the `securewave` directory. If you attempt to extract previously downloaded encrypted files (including Signal Studio downloaded files or internally created signal generator files (ESG/PSG only)) *without* using the securewave directory, the signal generator generates an error and displays:<br>`ERROR: 221, Access Denied.` |

Encrypted I/Q Files and the Securewave Directory (Keysight MXG)

| | |
|---|---|
| **NOTE** | Header parameters of files stored on the Keysight MXG's internal or USB media cannot be changed unless the file is copied to the volatile BBG memory. For more information on modifying header parameters, refer to the **User's Guide**. |

When downloading encrypted files (`.SECUREWAVE`) from the USB media that have had the file suffix changed to something other than `.SECUREWAVE`, you must use the **Use As** or **Copy File to Instrument** menus to play an encrypted waveform file in the signal generator.

## File Transfer Methods

— SCPI using VXI–11 (VMEbus Extensions for Instrumentation as defined in VXI–11)
— SCPI over the GPIB or RS 232
— SCPI with sockets LAN (using port 5025)
— File Transfer Protocol (FTP)

## SCPI Command Line Structure

The signal generator expects to see waveform data as block data (binary files). The IEEE standard 488.2–1992 section 7.7.6 defines block data. The following example shows how to structure a SCPI command for downloading waveform data (#ABC represents the block data):

```
:MMEM:DATA "<file_name>",#ABC
```

| | |
|---|---|
| "<file_name>" | the I/Q file name and file path within the signal generator |
| # | indicates the start of the data block |
| A | the number of decimal digits present in B |
| B | a decimal number specifying the number of data bytes to follow in C |
| C | the actual binary waveform data |

The following example demonstrates this structure:

```
MMEM:DATA "WFM1:my_file",#3 240 12%S!4&07#8g*Y9@7...
```

file_name            A  B            C

| | |
|---|---|
| WFM1: | the file path |
| my_file | the I/Q file name as it will appear in the signal generator's memory catalog |
| # | indicates the start of the data block |
| 3 | B has three decimal digits |
| 240 | 240 bytes of data to follow in C |
| 12%S!4&07#8g*Y9@7... | the ASCII representation of some of the binary data downloaded to the signal generator, however not all ASCII values are printable |

## Commands and File Paths for Downloading and Extracting Waveform Data

| NOTE | Filenames should **not** exceed 23 characters. |
|---|---|

You can download or extract waveform data using the commands and file paths in the following tables:

— Table 7, "Downloading Unencrypted Files for No Extraction (Extraction allowed on the Keysight MXG Only)," on page 30

Table 7        Downloading Unencrypted Files for No Extraction (Extraction allowed on the Keysight MXG[1] Only)

| Download Method/ Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile memory | `MMEM:DATA "WFM1:<file_name>",<blockdata>`<br>`MMEM:DATA "MKR1:<file_name>",<blockdata>`<br>`MMEM:DATA "HDR1:<file_name>",<blockdata>` |
| SCPI/volatile memory with full directory path | `MMEM:DATA "user/bbg1/waveform/<file_name>",<blockdata>`<br>`MMEM:DATA "user/bbg1/markers/<file_name>",<blockdata>`<br>`MMEM:DATA "user/bbg1/header/<file_name>",<blockdata>` |
| SCPI/non–volatile memory | `MMEM:DATA "NVWFM:<file_name>",<blockdata>`<br>`MMEM:DATA "NVMKR:<file_name>",<blockdata>`<br>`MMEM:DATA "NVHDR:<file_name>",<blockdata>` |
| SCPI/non–volatile memory with full directory path | `MMEM:DATA /user/waveform/<file_name>",<blockdata>`<br>`MMEM:DATA /user/markers/<file_name>",<blockdata>`<br>`MMEM:DATA /user/header/<file_name>",<blockdata>` |

1. Refer to note on page 26.

Table 8        Downloading Encrypted Files for No Extraction (Extraction allowed on the Keysight MXG[1] Only)

| Download Method /Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile memory | `MMEM:DATA "user/bbg1/securewave/<file_name>",<blockdata>`<br>`MMEM:DATA "SWFM1:<file_name>",<blockdata>`<br>`MMEM:DATA "file_name@SWFM1",<blockdata>` |
| SCPI/non–volatile memory | `MMEM:DATA "user/securewave/<file_name>",<blockdata>`<br>`MMEM:DATA "SNVWFM:<file_name>",<blockdata>`<br>`MMEM:DATA "file_name@SNVWFM",<blockdata>` |

1. Refer to note on page 26.

Table 9                    Downloading Unencrypted Files for Extraction

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile memory[1] | `MEM:DATA:UNPRotected "/user/bbg1/waveform/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "/user/bbg1/markers/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "/user/bbg1/header/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "WFM1:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "MKR1:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "HDR1:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@WFM1",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@MKR1",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@HDR1",<blockdata>` |
| SCPI/non–volatile memory[1] | `MEM:DATA:UNPRotected "/user/waveform/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "/user/markers/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "/user/header/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "NVWFM:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "NVMKR:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "NVHDR:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@NVWFM",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@NVMKR",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@NVHDR",<blockdata>` |
| FTP/volatile memory[2] | `put <file_name> /user/bbg1/waveform/<file_name>`<br>`put <file_name> /user/bbg1/markers/<file_name>`<br>`put <file_name> /user/bbg1/header/<file_name>` |
| FTP/non–volatile memory[2] | `put <file_name> /user/waveform/<file_name>`<br>`put <file_name> /user/markers/<file_name>`<br>`put <file_name> /user/header/<file_name>` |

1. On the N5162A/82A the `:MEM:DATA:UNPRotected` command is **not** required to be able to extract files (i.e. use `:MEM:DATA`). For more information, refer to the **SCPI Command Reference**.
2. See "FTP Procedures" on page 34.

Table 10                    Extracting Unencrypted I/Q Data

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile memory | `MMEM:DATA? "/user/bbg1/waveform/<file_name>"`<br>`MMEM:DATA? "WFM1:<file_name>"`<br>`MMEM:DATA? "<file_name>@WFM1"` |
| SCPI/non–volatile memory | `MMEM:DATA? "/user/waveform/<file_name>"`<br>`MMEM:DATA? "NVWFM:<file_name>"`<br>`MMEM:DATA? "<file_name>@NVWFM"` |
| FTP/volatile memory[1] | `get /user/bbg1/waveform/<file_name>`<br>`get /user/bbg1/markers/<file_name>`<br>`get /user/bbg1/header/<file_name>` |

**Table 10          Extracting Unencrypted I/Q Data**

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| FTP/non–volatile memory[1] | `get /user/waveform/<file_name>`<br>`get /user/markers/<file_name>`<br>`get /user/header/<file_name>` |

1. See "FTP Procedures" on page 34.

**Table 11          Downloading Encrypted Files for Extraction**

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile[1] memory | `MEM:DATA:UNPRotected "/user/bbg1/securewave/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "SWFM1:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@SWFM1",<blockdata>` |
| SCPI/non–volatile memory<br>SCPI/non–volatile memory[1] | `MEM:DATA:UNPRotected "/user/securewave/file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "SNVWFM:file_name",<blockdata>`<br>`MEM:DATA:UNPRotected "file_name@SNVWFM",<blockdata>` |
| FTP/volatile memory[2] | `put <file_name> /user/bbg1/securewave/<file_name>` |
| FTP/non–volatile memory[2] | `put <file_name> /user/securewave/<file_name>` |

1. On the N5162A/82A the `:MEM:DATA:UNPRotected` command is **not** required to be able to extract files (i.e. use `:MEM:DATA`). For more information, refer to the **SCPI Command Reference**.
2. See "FTP Procedures" on page 34.

**Table 12          Extracting Encrypted Waveform Data**

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| SCPI/volatile memory | `MMEM:DATA? "/user/bbg1/securewave/file_name"`<br>`MMEM:DATA? "SWFM1:file_name"`<br>`MMEM:DATA? "file_name@SWFM1"` |
| SCPI/non–volatile memory | `MMEM:DATA? "/user/securewave/file_name"`<br>`MMEM:DATA? "SNVWFM:file_name"`<br>`MMEM:DATA? "file_name@SNVWFM"` |
| FTP/volatile memory[1] | `get /user/bbg1/securewave/<file_name>` |

Table 12          Extracting Encrypted Waveform Data

| Download Method/Memory Type | Command Syntax Options |
|---|---|
| FTP/non–volatile memory[1] | `get /user/securewave/<file_name>` |

1. See "FTP Procedures" on page 34.

## FTP Procedures

**CAUTION**  Avoid using the *OPC? or *WAI commands to verify that the FTP process has been completed. These commands can potentially hang up due to the processing of other SCPI parser operations. Refer to the **SCPI Command Reference**.

**NOTE**  If you are remotely FTPing files and need to verify the completion of the FTP process, then query the instrument by using SCPI commands such as: ':MEM:DATA:', ':MEM:CAT', '*STB?', 'FREQ?', '*IDN?', 'OUTP:STAT?'. Refer to the **SCPI Command Reference**.

There are three ways to FTP files:

— use Microsoft's® Internet Explorer FTP feature
— use the PC's or UNIX command window
— use the signal generator's internal web server following the firmware requirements in the table below

| Signal Generator | Firmware Version (Required for Web Server Compatibility) |
|---|---|
| N516xA[1], N518xA | All |
| E44x8C | $\geq$ C.03.10 |
| E82x7D, E8663B | All |

1. The N5162A requires firmware version A.0140 or newer.

### Using Microsoft's Internet Explorer

1. Enter the signal generator's hostname or IP address as part of the FTP URL.

   *ftp://<host name> or*

   **ftp://<IP address>**

2. Press **Enter** on the keyboard or **Go** from the Internet Explorer window.

   The signal generator files appear in the Internet Explorer window.

3. Drag and drop files between the PC and the Internet Explorer window

## Using the Command Window (PC or UNIX)

This procedure downloads to non–volatile memory. To download to volatile memory, change the file path.

| | |
|---|---|
| **CAUTION** | Get and Put commands write over existing files by the same name in destination directories. Remember to change remote and local filenames to avoid the loss of data. |

| | |
|---|---|
| **NOTE** | If a filename has a space, quotations are required around the filename. |
| | Always transfer the waveform file before transferring the marker file. |
| | For additional information on FTP commands, refer to the operating system's **Window Help and Support Center**. |

1. From the PC command prompt or UNIX command line, change to the destination directory for the file you intend to download.

2. From the PC command prompt or UNIX command line, type `ftp <instrument name>`. Where `instrument name` is the signal generator's hostname or IP address.

3. At the `User:` prompt in the ftp window, press **Enter** (no entry is required).

4. At the `Password:` prompt in the ftp window, press **Enter** (no entry is required).

5. At the `ftp` prompt, either **put** a file or **get** a file:

   **To put a file, type:**

   `put <file_name> /user/waveform/<file_name1>`

   where `<file_name>` is the name of the file to download and `<file_name1>` is the name designator for the signal generator's `/user/waveform/` directory.

   If `<filename1>` is unspecified, ftp uses the specified `<file_name>` to name `<file_name1>`.

   — If a marker file is associated with the data file, use the following command to download it to the signal generator:
   `put <marker file_name> /user/markers/<file_name1>`

   where <marker file_name> is the name of the file to download and <file_name1> is the name designator for the file in the signal generator's `/user/markers/` directory. Marker files and the associated I/Q waveform data have the same name.

For more examples of put command usage refer to Table 13.

Table 13           Put Command Examples

| Command Results | Local | Remote | Notes |
|---|---|---|---|
| **Incorrect** | `put <filename.wfm>`<br><br>`put <filename.mkr>` | `/user/waveform/<filename 1.wfm>`<br><br>`/user/marker/<filename1. mkr>` | Produces two separate and incompatible files. |
| **Correct** | `put <filename.wfm>`<br><br>`put <filename.mkr>` | `/user/waveform/<filename 1>`<br><br>`/user/marker/<filename1>` | Creates a waveform file and a compatible marker file. |

**To get a file, type:**

`get /user/waveform/<file_name1> <file_name>`

where `<file_name1>` is the file to download from the signal generator's `/user/waveform/` directory and `<file_name>` is the name designator for the local PC/UNIX.

— If a marker file is associated with the data file, use the following command to download it to the local PC/UNIX directory:
`get /user/markers/<file_name1> <marker file_name>`

where `<marker file_name1>` is the name of the marker file to download from the signal generator's `/user/markers/` directory and `<marker file_name>` is the name of the file to be downloaded to the local PC/UNIX.

For more examples of get command usage refer to Table 14.

Table 14           Get Command Examples

| Command Results | Local | Remote | Notes |
|---|---|---|---|
| **Incorrect** | `get /user/waveform/file`<br><br>`get /user/marker/file` | `file1`<br>`file1` | Results in file1 containing only the marker data. |
| **Correct** | `get /user/waveform/file`<br><br>`get /user/marker/file` | `file1.wfm`<br>`file1.mkr` | Creates a waveform file and a compatible marker file. It is easier to keep files associated by varying the extenders. |

6. At the `ftp` prompt, type: `bye`

7. At the command prompt, type: `exit`

### Using the Signal Generator's Internal Web Server

1. Enter the signal generator's hostname or IP address in the URL.

   *http://<host name> or <IP address>*

2. Click the **Signal Generator FTP Access** button located on the left side of the window.

   The signal generator files appear in the web browser's window.

3. Drag and drop files between the PC and the browser's window

For more information on the web server feature, see **Programming Guide**.

# Creating Waveform Data

This section examines the C++ code algorithm for creating I/Q waveform data by breaking the programming example into functional parts and explaining the code in generic terms. This is done to help you understand the code algorithm in creating the I and Q data, so you can leverage the concept into your programming environment. The **SCPI Command Reference**, contains information on how to use SCPI commands to define the markers (polarity, routing, and other marker settings). If you do not need this level of detail, you can find the complete programming examples in "Programming Examples" on page 57.

You can use various programming environments to create ARB waveform data. Generally there are two types:

— **Simulation software**— this includes MATLAB, Keysight Technologies EESof Advanced Design System (ADS), Signal Processing WorkSystem (SPW), and so forth.

— **Advanced programming languages**—this includes, C++, VB, VEE, MS Visual Studio.Net, Labview, and so forth.

No matter which programming environment you use to create the waveform data, make sure that the data conforms to the data requirements shown on page 4. To learn about I/Q data for the signal generator, see "Understanding Waveform Data" on page 5.

## Code Algorithm

This section uses code from the C++ programming example "Importing, Byte Swapping, Interleaving, and Downloading I and Q Data—Big and Little Endian Order" on page 79 to demonstrate how to create and scale waveform data.

There are three steps in the process of creating an I/Q waveform:

1. Create the I and Q data.
2. Save the I and Q data to a text file for review.
3. Interleave the I and Q data to make an I/Q file, and swap the byte order for little–endian platforms.

For information on downloading I/Q waveform data to a signal generator, refer to "Commands and File Paths for Downloading and Extracting Waveform Data" on page 29 and "Downloading Waveform Data" on page 44.

### 1. Create I and Q data.

The following lines of code create scaled I and Q data for a sine wave. The I data consists of one period of a sine wave and the Q data consists of one period of a cosine wave.

| Line | Code—Create I and Q data |
|------|--------------------------|

```
1    const int NUMSAMPLES=500;
2    main(int argc, char* argv[]);
3    {
4        short idata[NUMSAMPLES];
5        short qdata[NUMSAMPLES];
6     int numsamples = NUMSAMPLES;
7        for(int index=0; index<numsamples; index++);
8         {
9            idata[index]=23000 * sin((2*3.14*index)/numsamples);
10           qdata[index]=23000 * cos((2*3.14*index)/numsamples);
11        }
```

| Line | Code Description—Create I and Q data |
|------|--------------------------------------|
| 1 | Define the number of waveform points. Note that the maximum number of waveform points that you can set is based on the amount of available memory in the signal generator. For more information on signal generator memory, refer to "Waveform Memory" on page 19. |
| 2 | Define the main function in C++. |
| 4 | Create an array to hold the generated I values. The array length equals the number of the waveform points. Note that we define the array as type **short**, which represents a 16–bit signed integer in most C++ compilers. |
| 5 | Create an array to hold the generated Q values (signed 16–bit integers). |
| 6 | Define and set a temporary variable, which is used to calculate the I and Q values. |
| 7–11 | Create a loop to do the following:<br><br>— Generate and scale the I data (DAC values). This example uses a simple sine equation, where 2*3.14 equals one waveform cycle. Change the equation to fit your application.<br><br>    — The array pointer, **index**, increments from 0–499, creating 500 I data points over one period of the sine waveform.<br><br>    — Set the scale of the DAC values in the range of -32768 to 32767, where the values -32768 and 32767 equal full scale negative and positive respectively. This example uses 23000 as the multiplier, resulting in approximately 70% scaling. For more information on scaling, see "Scaling DAC Values" on page 9.<br><br>**NOTE** The signal generator comes from the factory with I/Q scaling set to 70%. If you reduce the DAC input values, ensure that you set the signal generator scaling (`:RADio:ARB:RSCaling`) to an appropriate setting that accounts for the reduced values.<br><br>— Generate and scale the Q data (DAC value). This example uses a simple cosine equation, where 2*3.14 equals one waveform cycle. Change the equation to fit your application.<br><br>    — The array pointer, **index**, increments from 0–499, creating 500 Q data points over one period of the cosine waveform.<br><br>    — Set the scale of the DAC values in the range of -32767 to 32768, where the values -32767 and 32768 equal full scale negative and positive respectively. This example uses 23000 as the multiplier, resulting in approximately 70% scaling. For more information on scaling, see "Scaling DAC Values" on page 9. |

## 2. Save the I/Q data to a text file to review.

The following lines of code export the I and Q data to a text file for validation. After exporting the data, open the file using Microsoft Excel or a similar spreadsheet program, and verify that the I and Q data are correct.

| Line | Code Description—Saving the I/Q Data to a Text File |
|---|---|

```
12    char *ofile = "c:\\temp\\iq.txt";
13    FILE *outfile = fopen(ofile, "w");
14    if (outfile==NULL) perror ("Error opening file to write");
15    for(index=0; index<numsamples; index++)
16    {
17        fprintf(outfile, "%d, %d\n", idata[index], qdata[index]);
18    }
19    fclose(outfile);
```

| Line | Code Description—Saving the I/Q Data to a Text File |
|---|---|
| 12 | Set the absolute path of a text file to a character variable. In this example, *iq.txt* **is the file name and** *\*ofile* **is the variable name**.<br><br>For the file path, some operating systems may not use the drive prefix ('c:' in this example), or may require only a single forward slash (/), or both ("/*temp*/*iq.txt*") |
| 13 | Open the text file in *write* format. |
| 14 | If the text file does not open, print an error message. |
| 15–18 | Create a loop that prints the array of generated I and Q data samples to the text file. |
| 19 | Close the text file. |

### 3. Interleave the I and Q data, and byte swap if using little endian order.

This step has two sets of code:

— Interleaving and byte swapping I and Q data for little endian order
— Interleaving I and Q data for big endian order

For more information on byte order, see "Little Endian and Big Endian (Byte Order)" on page 6.

| Line | Code—Interleaving and Byte Swapping for Little Endian Order |
|------|-------------------------------------------------------------|

```
20    char iqbuffer[NUMSAMPLES*4];
21    for(index=0; index<numsamples; index++)
22    {
23        short ivalue = idata[index];
24        short qvalue = qdata[index];
25        iqbuffer[index*4]   = (ivalue >> 8) & 0xFF;
26        iqbuffer[index*4+1] = ivalue & 0xFF;
27        iqbuffer[index*4+2] = (qvalue >> 8) & 0xFF;
28        iqbuffer[index*4+3] = qvalue & 0xFF;
29    }
30    return 0;
```

| Line | Code Description—Interleaving and Byte Swapping for Little Endian Order |
|------|------------------------------------------------------------------------|
| 20 | Define a character array to store the interleaved I and Q data. The character array makes byte swapping easier, since each array location accepts only 8 bits (1 byte). The array size increases by four times to accommodate two bytes of I data and two bytes of Q data. |
| 21–29 | Create a loop to do the following:<br><br>— Save the current I data array value to a variable.<br>**NOTE** In rare instances, a compiler may define *short* as larger than 16 bits. If this condition exists, replace *short* with the appropriate object or label that defines a 16–bit integer.<br>— Save the current Q data array value to a variable.<br>— Swap the low bytes (bits 0–7) of the data with the high bytes of the data (done for both |

| Line | Code Description—Interleaving and Byte Swapping for Little Endian Order |
|------|---------------------------------------------------------------------------|
| 21–29 | the I and Q data), and interleave the I and Q data.<br><br>— shift the data pointer right 8 bits to the beginning of the high byte (*ivalue >> 8*)<br><br>**Little Endian Order**<br><br>7 6 5 4 3 2 1 0   15 14 13 12 11 10 9 8   Bit Position<br>1 1 1 0 1 0 0 1   1 0 1 1 0 1 1 1   Data<br><br>Hex values = E9 B7<br><br>Data pointer        Data pointer shifted 8 bits<br><br>— *AND* (boolean) the high I byte with 0xFF to make the high I byte the value to store in the IQ array—(**ivalue >> 8**) *& 0xFF*<br><br>15 14 13 12 11 10 9 8<br>1 0 1 1 0 1 1 1   Hex value =B7<br>1 1 1 1 1 1 1 1   Hex value =FF<br>1 0 1 1 0 1 1 1   Hex value =B7<br><br>— *AND* (boolean) the low I byte with 0xFF (*ivalue & 0xFF*) to make the low I byte the value to store in the I/Q array location just after the high byte [*index * 4 + 1*]<br><br>**I Data in I/Q Array after Byte Swap (Big Endian Order)**<br><br>15 14 13 12 11 10 9 8   7 6 5 4 3 2 1 0   Bit Position<br>1 0 1 1 0 1 1 1   1 1 1 0 1 0 0 1   Data<br>Hex value = B7 E9<br><br>— Swap the Q byte order within the same loop. Notice that the I and Q data interleave with each loop cycle. This is due to the I/Q array shifting by one location for each I and Q operation [*index * 4 + n*].<br><br>**Interleaved I/Q Array in Big Endian Order**<br><br>15.................... 8 7................... 0  15.................... 8 7................... 0  Bit Position<br>1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 1   1 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1   Data<br><br>I Data                               Q Data |

| Line | Code—Interleaving I and Q data for Big Endian Order |

```
20    short iqbuffer[NUMSAMPLES*2];
21    for(index=0; index<numsamples; index++)
22    {
23    iqbuffer[index*2]   = idata[index];
24    iqbuffer[index*2+1] = qdata[index];
25    }
26    return 0;
```

| Line | Code Description—Interleaving I and Q data for Big Endian Order |
|------|------------------------------------------------------------------|
| 20 | Define a 16–bit integer (short) array to store the interleaved I and Q data. The array size increases by two times to accommodate two bytes of I data and two bytes of Q data.<br><br>**NOTE**     In rare instances, a compiler may define *short* as larger than 16 bits. If this condition exists, replace *short* with the appropriate object or label that defines a 16–bit integer. |
| 21–25 | Create a loop to do the following:<br><br>— Store the I data values to the I/Q array location [*index*\*2].<br>— Store the Q data values to the I/Q array location [*index*\*2+1].<br><br>**Interleaved I/Q Array in Big Endian Order**<br><br>15..................... 8 7.................... 0   15..................... 8 7.................... 0   Bit Position<br>1 0 1 1 0 1 1 1 1 1 1 0 1 0 0 1   1 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1   Data<br><br>         I Data                Q Data |

To download the data created in the above example, see "Using Advanced Programming Languages" on page 46.

# Downloading Waveform Data

This section examines methods of downloading I/Q waveform data created in MATLAB (a simulation software) and C++ (an advanced programming language). For more information on simulation and advanced programming environments, see "Creating Waveform Data" on page 38.

To download data from simulation software environments, it is typically easier to use one of the free download utilities (described on page 53), because simulation software usually saves the data to a file. In MATLAB however, you can either save data to a .mat file or create a complex array. To facilitate downloading a MATLAB complex data array, Keysight created the Waveform Download Assistant (one of the free download utilities), which downloads the complex data array from within the MATLAB environment. This section shows how to use the Waveform Download Assistant.

For advanced programming languages, this section closely examines the code algorithm for downloading I/Q waveform data by breaking the programming examples into functional parts and explaining the code in generic terms. This is done to help you understand the code algorithm in downloading the interleaved I/Q data, so you can leverage the concept into your programming environment. While not discussed in this section, you may also save the data to a binary file and use one of the download utilities to download the waveform data (see "Using the Download Utilities" on page 53).

If you do not need the level of detail this section provides, you can find complete programming examples in "Programming Examples" on page 57. Prior to downloading the I/Q data, ensure that it conforms to the data requirements shown on page 4. To learn about I/Q data for the signal generator, see "Understanding Waveform Data" on page 5. For creating waveform data, see "Creating Waveform Data" on page 38.

**NOTE**    To avoid overwriting the current waveform in volatile memory, before downloading files into volatile memory (WFM1), change the file name or turn off the ARB. For more information, on manually turning off the ARB, refer to the **User's Guide**.

To turn off the ARB remotely, send: `:SOURce:RADio:ARB:STATe OFF`.

## Using Simulation Software

This procedure uses a complex data array created in MATLAB and uses the Waveform Download Assistant to download the data. To obtain the Waveform Download Assistant, see "Using the Download Utilities" on page 53.

There are two steps in the process of downloading an I/Q waveform:

1. Open a connection session.
2. Download the I/Q data.

## 1. Open a connection session with the signal generator.

The following code establishes a LAN connection with the signal generator, sends the IEEE SCPI command `*idn?`, and if the connection fails, displays an error message.

**Line**     **Code—Open a Connection Session**

```
1    io = agt_newconnection('tcpip','IP address');
     %io = agt_newconnection('gpib',<primary address>,<secondary
2    address>);
3    [status,status_description,query_result] = agt_query(io,'*idn?');
4    if status == -1
5    display 'fail to connect to the signal generator';
     end;
```

| Line | Code Description—Open a Connection Session with the Signal Generator |
|------|----------------------------------------------------------------------|
| 1 | Sets up a structure (indicated above by **io**) used by subsequent function calls to establish a LAN connection to the signal generator. <br><br> — **agt_newconnection()** is the function of Waveform Download Assistant used in MATLAB to build a connection to the signal generator. <br><br> — If you are using GPIB to connect to the signal generator, provide the board, primary address, and secondary address: **io = agt_newconnection('gpib',0,19);** <br> Change the GPIB address based on your instrument setting. |
| 2 | Send a query to the signal generator to verify the connection. <br><br> — **agt_query()** is an Waveform Download Assistant function that sends a query to the signal generator. <br><br> — If signal generator receives the query `*idn?`, *status* returns zero and *query_result* returns the signal generator's model number, serial number, and firmware version. |
| 3–5 | If the query fails, display a message. |

## 2. Download the I/Q data

The following code downloads the generated waveform data to the signal generator, and if the download fails, displays a message.

**Line**     **Code—Download the I/Q data**

```
6    [status, status_description] = agt_waveformload(io, IQwave,
     'waveformfile1', 2000, 'no_play','norm_scale');
7    if status == -1
8    display 'fail to download to the signal generator';
9    end;
```

| Line | Code Description—Download the I/Q data |
|------|----------------------------------------|
| 6 | Download the I/Q waveform data to the signal generator by using the function call (*agt_waveformload*) from the Waveform Download Assistant. Some of the arguments are optional as indicated below, but if one is used, you must use all arguments previous to the one you require.<br><br>Notice that with this function, you can perform the following actions:<br><br>— download complex I/Q data<br>— name the file (optional argument)<br>— set the sample rate (optional argument)<br>    If you do not set a value, the signal generator uses its preset value of 125 MHz (N5162A/82A) or 100 MHz (E4438C/E8267D), or if a waveform was previously play, the value from that waveform.<br>— start or not start waveform playback after downloading the data (optional argument)<br>    Use either the argument *play* or the argument **no_play**.<br>— whether to normalize and scale the I/Q data (optional argument)<br>    If you normalize and scale the data within the body of the code, then use **no_normscale**, but if you need to normalize and scale the data, use *norm_scale*. This normalizes the waveform data to the DAC values and then scales the data to 70% of the DAC values.<br>— download marker data (optional argument)<br>    If there is no marker data, the signal generator creates a default marker file, all marker set to zero.<br><br>To verify the waveform data download, see "Loading, Playing, and Verifying a Downloaded Waveform" on page 50. |
| 7–9 | If the download fails, display an error message. |

## Using Advanced Programming Languages

This procedure uses code from the C++ programming example "Importing, Byte Swapping, Interleaving, and Downloading I and Q Data—Big and Little Endian Order" on page 79.

For information on creating I/Q waveform data, refer to "Creating Waveform Data" on page 38.

There are two steps in the process of downloading an I/Q waveform:

1. Open a connection session.
2. Download the I/Q data.

### 1. Open a connection session with the signal generator.

The following code establishes a LAN connection with the signal generator or prints an error message if the session is not opened successfully.

| Line | Code Description—Open a Connection Session |
|------|---------------------------------------------|

```
1    char* instOpenString ="lan[hostname or IP address]";
     //char* instOpenString ="gpib<primary addr>,<secondary addr>";
2    INST id=iopen(instOpenString);
3    if (!id)
4    {
5        fprintf(stderr, "iopen failed (%s)\n", instOpenString);
6        return -1;
7    }
```

| Line | Code Description—Open a Connection Session |
|------|---------------------------------------------|
| 1 | Assign the signal generator's LAN hostname, IP address, or GPIB address to a character string.<br><br>— This example uses the IO library's *iopen()* SICL function to establish a LAN connection with the signal generator. The input argument, **lan[hostname or IP address]** contains the device, interface, or commander address. Change it to your signal generator host name or just set it to the IP address used by your signal generator. For example: "*lan[999.137.240.9]*"<br><br>— If you are using GPIB to connect to the signal generator, use the commented line in place of the first line. Insert the GPIB address based on your instrument setting, for example "*gpib0,19*".<br><br>— For the detailed information about the parameters of the SICL function *iopen()*, refer to the online "*Keysight SICL User's Guide for Windows.*" |
| 2 | Open a connection session with the signal generator to download the generated I/Q data.<br><br>The SICL function *iopen()* is from the IO library and creates a session that returns an identifier to *id*.<br><br>— If *iopen()* succeeds in establishing a connection, the function returns a valid session *id*. The valid session *id* is not viewable, and can only be used by other SICL functions.<br><br>— If *iopen()* generates an error before making the connection, the session identifier is always set to zero. This occurs if the connection fails.<br><br>— To use this function in C++, you must include the standard header #include <sicl.h> before the main() function. |
| 3–7 | If *id* = 0, the program prints out the error message and exits the program. |

### 2. Download the I/Q data.

The following code sends the SCPI command and downloads the generated waveform data to the signal generator.

| Line | CodeDescription—Download the I/Q Data |
|---|---|

```
8    int bytesToSend;
9    bytesToSend = numsamples*4;
10   char s[20];
11   char cmd[200];
12   sprintf(s, "%d", bytesToSend);
13   sprintf(cmd, ":MEM:DATA \"WFM1:FILE1\", #%d%d", strlen(s),
     bytesToSend);
14   iwrite(id, cmd, strlen(cmd), 0, 0);
15   iwrite(id, iqbuffer, bytesToSend, 0, 0);
16   iwrite(id, "\n", 1, 1, 0);
```

| Line | Code Description—Download the I/Q data |
|---|---|
| 8 | Define an integer variable (*bytesToSend*) to store the number of bytes to send to the signal generator. |
| 9 | Calculate the total number of bytes, and store the value in the integer variable defined in line 8.<br><br>In this code, *numsamples* contains the number of waveform points, not the number of bytes. Because it takes four bytes of data, two I bytes and two Q bytes, to create one waveform point, we have to multiply *numsamples* by four. This is shown in the following example:<br><br>numsamples = 500 waveform points<br>numsamples x 4 = 2000 (four bytes per point)<br>bytesToSend = 2000 (numsamples x 4)<br><br>For information on setting the number of waveform points, see "1. Create I and Q data." on page 38. |
| 10 | Create a string large enough to hold the *bytesToSend* value as characters. In this code, string *s* is set to 20 bytes (20 characters—one character equals one byte) |
| 11 | Create a string and set its length (*cmd*[200]) to hold the SCPI command syntax and parameters. In this code, we define the string length as 200 bytes (200 characters). |
| 12 | Store the value of *bytesToSend* in string *s*. For example, if bytesToSend = 2000; *s* = "2000"<br><br>**sprintf()** is a standard function in C++, which writes string data to a string variable. |
| 13 | Store the SCPI command syntax and parameters in the string **cmd**. The SCPI command prepares the signal generator to accept the data.<br><br>— **strlen()** is a standard function in C++, which returns length of a string.<br><br>— If *bytesToSend* = 2000, then *s* = "2000", *strlen(s)* = 4, so<br>*cmd* = :MEM:DATA "WFM1:FILE1\" #42000. |
| 14 | Send the SCPI command stored in the string *cmd* to the signal generator, which is represented by the session *id*.<br><br>— *iwrite()* is a SICL function in IO library, which writes the data (block data) specified in the string *cmd* to the signal generator (*id*).<br><br>— The third argument of *iwrite()*, *strlen(cmd)*, informs the signal generator of the number of bytes in the command string. The signal generator parses the string to determine the number of I/Q data bytes it expects to receive.<br><br>— The fourth argument of iwrite(), 0, means there is no END of file indicator for the string. This lets the session remain open, so the program can download the I/Q data. |

| Line | Code Description—Download the I/Q data |
|---|---|
| 15 | Send the generated waveform data stored in the I/Q array **(iqbuffer)** to the signal generator.<br><br>— *iwrite()* sends the data specified in *iqbuffer* to the signal generator (session identifier specified in *id*).<br><br>— The third argument of iwrite(), *bytesToSend*, contains the length of the *iqbuffer* in bytes. In this example, it is 2000.<br><br>— The fourth argument of *iwrite()*, 0, means there is no END of file indicator in the data.<br><br>In many programming languages, there are two methods to send SCPI commands and data:<br><br>— Method 1 where the program stops the data download when it encounters the first zero (END indicator) in the data.<br><br>— Method 2 where the program sends a fixed number of bytes and ignores any zeros in the data. This is the method used in our program.<br><br>For your programming language, you must find and use the equivalent of method two. Otherwise you may only achieve a partial download of the I and Q data. |
| 16 | Send the terminating carriage (\n) as the last byte of the waveform data.<br><br>— *iwrite()* writes the data "\n" to the signal generator (session identifier specified in *id*).<br><br>— The third argument of *iwrite()*, 1, sends one byte to the signal generator.<br><br>— The fourth argument of *iwrite()*, 1, is the END of file indicator, which the program uses to terminate the data download.<br><br>To verify the waveform data download, see "Loading, Playing, and Verifying a Downloaded Waveform" on page 50. |

## Loading, Playing, and Verifying a Downloaded Waveform

The following procedures show how to perform the steps using SCPI commands. For front panel key commands, refer to the **User's Guide** or to the Key help in the signal generator.

### Loading a File from Non–Volatile Memory

Select the downloaded I/Q file in non–volatile waveform memory (NVWFM) and load it into volatile waveform memory (WFM1). The file comprises three items: I/Q data, marker file, and file header information.

Send one of the following SCPI command to copy the I/Q file, marker file and file header information:

```
:MEMory:COPY:NAME "<NVWFM:file_name>","<WFM1:file_name>"
:MEMory:COPY:NAME "<NVMKR:file_name>","<MKR1:file_name>"
:MEMory:COPY:NAME "<,"<HDR:file_name>"
```

**NOTE**    When you copy a waveform file, marker file, or header file information from volatile or non–volatile memory, the waveform and associated marker and header files are all copied. Conversely, when you delete an I/Q file, the associated marker and header files are deleted. It is not necessary to send separate commands to copy or delete the marker and header files.

### Playing the Waveform

**NOTE**    If you would like to build and play a waveform **sequence**, refer to .

Play the waveform and use it to modulate the RF carrier.

1. List the waveform files from the volatile memory waveform list:

   Send the following SCPI command:

   ```
   :MMEMory:CATalog? "WFM1:"
   ```

2. Select the waveform from the volatile memory waveform list:

   Send the following SCPI command:

   ```
   :SOURce:RADio:ARB:WAVeform "WFM1:<file_name>"
   ```

3. Play the waveform:

   Send the following SCPI commands:

```
:SOURce:RADio:ARB:STATe ON
:OUTPut:MODulation:STATe ON
:OUTPut:STATe ON
```

## Verifying the Waveform

Perform this procedure after completing the steps in the previous procedure, .

**1.** Connect the signal generator to an oscilloscope as shown in the figure.
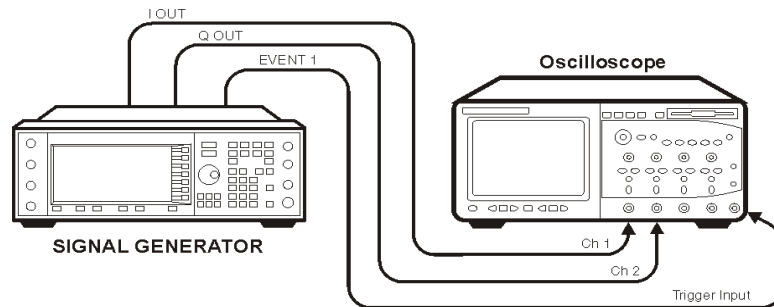


**2.** Set an active marker point on the first waveform point for marker one.

**NOTE**    Select the same waveform selected in .

Send the following SCPI commands:

```
:SOURce:RADio:ARB:MARKer:CLEar:ALL "WFM1:<file_name>",1
```

```
:SOURce:RADio:ARB:MARKer:SET "WFM1:<file_name>",1,1,1,0.
```

**3.** Compare the oscilloscope display to the plot of the I and Q data from the text file you created when you generated the data.

If the oscilloscope display, and the I and Q data plots differ, recheck your code. For detailed information on programmatically creating and downloading waveform data, see and . For information on the waveform data requirements, see .

## Building and Playing Waveform Sequences

The signal generator can be used to build waveform sequences. This section assumes you have created the waveform segment file(s) and have the waveform segment file(s) in volatile memory. The following SCPI commands can be used to generate and work with a waveform sequence. For more information refer to the signal generator's **SCPI Command Reference** and **User's Guide**.

NOTE    If you would like to verify the waveform sequence, refer to "Verifying the Waveform" on page 51.

1. List the waveform files from the volatile memory waveform list:

   Send the following SCPI command:

   ```
   :MMEMory:CATalog? "WFM1:"
   ```

2. Select the waveform segment file(s) from the volatile memory waveform list:

   Send the following SCPI command:

   ```
   :SOURce:RADio:ARB:WAVeform "WFM1:<file_name>"
   ```

3. Save the waveform segment(s) ("<waveform1>", "<waveform2>", ...), to non–volatile memory as a waveform sequence ("<file_name>"), define the number of repetitions (<reps>), each waveform segment plays, and enable/disable markers (M1|M2|M3|M4|...), for each waveform segment:

   Send the following SCPI command:

   ```
   :SOURce:RADio:ARB:SEQuence
   "<file_name>","<waveform1>",<reps>,M1|M2|M3|M4,{"<waveform
   2>",<reps>,ALL}
   ```

   ```
   :SOURce:RADio:ARB:SEQuence? "<file_name>"
   ```

NOTE    M1|M2|M3|M4 represent the number parameter of the marker selected (i.e. 1|2|3|4). Entering M1|M2|M3|M4 causes the signal generator to display an error. For more information on this SCPI command, refer to the signal generator's **SCPI Command Reference**.

4. Play the waveform sequence:

   Send the following SCPI commands:

   ```
   :SOURce:RADio:ARB:STATe ON
   :OUTPut:MODulation:STATe ON
   :OUTPut:STATe ON
   ```

# Using the Download Utilities

Keysight provides free download utilities to download waveform data into the signal generator. The table in this section describes the capabilities of three such utilities.

For more information and to install the utilities, refer to the following URLs:

— Keysight Signal Studio Toolkit 2:
*http://www.keysight.com/find/signalstudio*

This software provides a graphical interface for downloading files.

— IntuiLink for Keysight PSG/ESG/E8663B Signal Generators:
*http://www.keysight.com/find/intuilink*

This software places icons in the Microsoft Excel and Word toolbar. Use the icons to connect to the signal generator and open a window for downloading files.

---

**NOTE**    Intuilink is **not** available for the Keysight MXG.

---

— Waveform Download Assistant:
*http://www.keysight.com/find/downloadassistant*

This software provides functions for the MATLAB environment to download waveform data.

| Features | Keysight Signal Studio Toolkit 2 | IntuiLink[1] | Waveform Download Assistant |
|---|---|---|---|
| Downloads encrypted waveform files | X | | |
| Downloads complex MATLAB waveform data | | | X |
| Downloads MATLAB files (.mat) | X | | |
| Downloads unencrypted interleaved 16–bit I/Q files [2] | X | X | |
| Interleaves and downloads earlier 14–bit E443xB I and Q files[2] | X | X | |
| Swaps bytes for little endian order | | X | |
| Manually select big endian byte order for 14–bit and 16–bit I/Q files | X | | |
| Downloads user–created marker files | X | X | X |
| Performs scaling | X | X | X |
| Starts waveform play back | X | | X |
| Sends SCPI Commands and Queries | X | | X |
| Builds a waveform sequence | X | | X |

1. Intuilink is **not** available for the Keysight MXG.
2. ASCII or binary format.

# Downloading E443xB Signal Generator Files

To download earlier E443xB model I and Q files, use the same SCPI commands as if downloading files to an E443xB signal generator. The signal generator automatically converts the E443xB files to the proper file format as described in "Waveform Structure" on page 13 and stores them in the signal generator's memory. This conversion process causes the signal generator to take more time to download the earlier file format. To minimize the time to convert earlier E443xB files to the proper file format, store E443xB file downloads to volatile memory, and then transfer them over to non–volatile (NVWFM) memory.

---

**NOTE**       **You cannot extract waveform data downloaded as E443xB files.**

---

## E443xB Data Format

The following diagram describes the data format for the E443xB waveform files. This file structure can be compared with the new style file format shown in "Waveform Structure" on page 13. If you create new waveform files for the signal generator, use the format shown in "Waveform Data Requirements" on page  4.
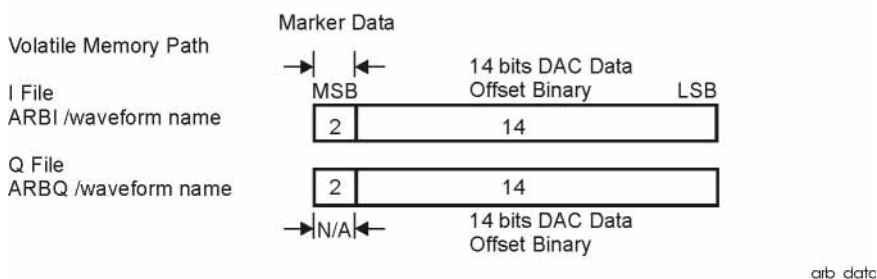


## Storage Locations for E443xB ARB files

Place waveforms in either volatile memory or non–volatile memory. The signal generator supports the E443xB directory structure for waveform file downloads (i.e. "ARBI:", "ARBQ:", "NVARBI:", and "NVARBQ:", see also "SCPI Commands" on page 56).

### Volatile Memory Storage Locations

— /user/arbi/
— /user/arbq/

### Non–Volatile Memory Storage Locations

— /user/nvarbi/

---

— /user/nvarbq/

Loading files into the above directories (volatile or non–volatile memory) does not actually store them in those directories. Instead, these directories function as "pipes" to the format translator. The signal generator performs the following functions on the E443xB data:

— Converts the 14–bit I and Q data into 16–bit data (the format required by the signal generator).
Subtract 8192, left shifts the data, and appends two bits (zeros) before the least significant bit (i.e. the offset binary values are converted to 2's complement values by the signal generator).

**E443xB 14–Bit Data**

I data                    Q data

1100110110111001    0000100111011001

Marker bits   14 data bits    Reserved bits   14 data bits
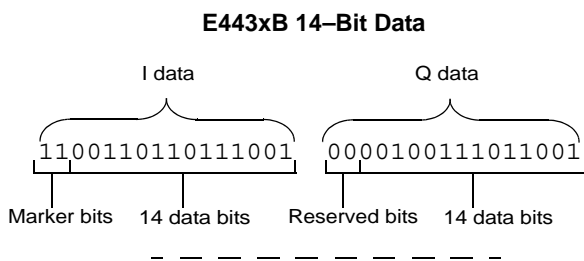
**Subtracts 8192, Left Shifts, and Adds Zeros—Removes Marker and Reserved Bits (16–Bit Data Format)**

16–bit I data                         16–bit Q data

11 1001011011100100            00 1000011101100100

Marker bits removed          Bits added          Reserved bits removed          Bits added
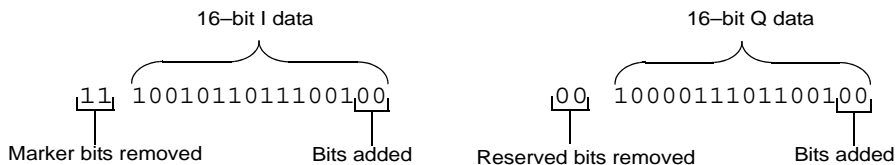
— Creates a marker file and places the marker information, bits 14 and 15 of the E443xB I data, into the marker file for markers one and two. Markers three and four, within the new marker file, are set to zero (off).

**Places the I Marker Bits into the Signal Generator Marker File**

0011

Marker 1 and 2 bits from the E443xB I data

Marker 3 and 4 bits
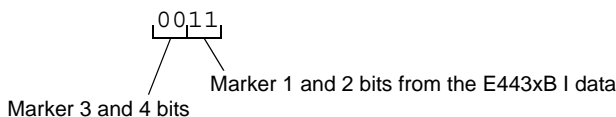
— Interleaves the 16–bit I and Q data creating one I/Q file.

— Creates a file header with all parameters set to unspecified (factory default file header setting).

## SCPI Commands

Use the following commands to download E443xB waveform files into the signal generator.

**NOTE**    To avoid overwriting the current waveform in volatile memory, before downloading files into volatile memory (`WFM1`), change the file name or turn off the ARB. For more information, on manually turning off the ARB, refer to the **User's Guide**.

To turn off the ARB remotely, send: `:SOURce:RADio:ARB:STATe OFF`.

| Extraction Method/ Memory Type | Command Syntax Options |
|---|---|
| SCPI/ volatile memory | `:MMEM:DATA "ARBI:<file_name>", <I waveform block data>` <br> `:MMEM:DATA "ARBQ:<file_name>", <Q waveform data>` |
| SCPI/ non–volatile memory | `:MMEM:DATA "NVARBI:<file_name>", <I waveform block data>` <br> `:MMEM:DATA "NVARBQ:<file_name>", <Q waveform block data>` |

The variables `<I waveform block data>` and `<Q waveform block data>` represents data in the E443xB file format. The string variable `<file_name>` is the name of the I and Q data file. After downloading the data, the signal generator associates a file header and marker file with the I/Q data file.

# Programming Examples

**NOTE**    The programming examples contain instrument–specific information. However, users can still use these programming examples by substituting in the instrument–specific information for your signal generator. Model specific exceptions for programming use, will be noted at the top of each programming section.

The programming examples use GPIB or LAN interfaces and are written in the following languages:

— C++ (page 58)
— MATLAB (page 87)
— Visual Basic (page 96)
— HP Basic (page 103)

See the signal generator's **Programming Guide** for information on interfaces and IO libraries.

The example programs are also available on the signal generator **Documentation CD–ROM**, which allows you to cut and paste the examples into an editor.

## C++ Programming Examples

This section contains the following programming examples:

### Creating and Storing Offset I/Q Data—Big and Little Endian Order

On the documentation CD, this programming example's name is "*offset_iq_c++.txt.*"

This C++ programming example (compiled using Microsoft Visual C++ 6.0) follows the same coding algorithm as the MATLAB programming example "Creating and Storing I/Q Data" on page  87 and performs the following functions:

— error checking
— data creation
— data normalization
— data scaling
— I/Q signal offset from the carrier (single sideband suppressed carrier signal)
— byte swapping and interleaving for little endian order data
— I and Q interleaving for big endian order data
— binary data file storing to a PC or workstation
— reversal of the data formatting process (byte swapping, interleaving, and normalizing the data)

After creating the binary file, you can use FTP, one of the download utilities, or one of the C++ download programming examples to download the file to the signal generator.

```
// This C++ example shows how to

// 1.) Create a simple IQ waveform

// 2.) Save the waveform into the ESG/PSG Internal Arb format

//      This format is for the E4438C, E8267C, E8267D

//      This format will not work with the ESG E443xB or the
Keysight MXG N518xA

// 3.) Load the internal Arb format file into an array
```

```
#include <stdio.h>

#include <string.h>

#include <math.h>


const int POINTS = 1000;   // Size of waveform

const char *computer = "PCWIN";


int main(int argc, char* argv[])

{


 // 1.) Create Simple IQ Signal
 ****************************************

        // This signal is a single tone on the upper

        // side of the carrier and is usually refered to as

        // a Single Side Band Suppressed Carrier (SSBSC) signal.

        // It is nothing more than a cosine wavefomm in I

        // and a sine waveform in Q.


        int points = POINTS; // Number of points in the waveform

        int cycles = 101; // Determines the frequency offset from the
carrier

        double Iwave[POINTS]; // I waveform

        double Qwave[POINTS]; // Q waveform

        short int waveform[2*POINTS]; // Holds interleaved I/Q data

        double maxAmp = 0; // Used to Normalize waveform data

        double minAmp = 0; // Used to Normalize waveform data

        double scale = 1;

        char buf; // Used for byte swapping

        char *pChar; // Used for byte swapping

        bool PC = true; // Set flag as appropriate


        double phaseInc = 2.0 * 3.141592654 * cycles / points;

        double phase = 0;

        int i = 0;
```

```
for( i=0; i<points; i++ )
{
  phase = i * phaseInc;
  Iwave[i] = cos(phase);
  Qwave[i] = sin(phase);
}


// 2.) Save waveform in internal format
*********************************
// Convert the I and Q data into the internal arb format
// The internal arb format is a single waveform containing
interleaved IQ
// data. The I/Q data is signed short integers (16 bits).
// The data has values scaled between +-32767 where
//   DAC Value   Description
//    32767      Maximum positive value of the DAC
//        0      Zero out of the DAC
//   -32767      Maximum negative value of the DAC
// The internal arb expects the data bytes to be in Big Endian
format.
// This is opposite of how short integers are saved on a PC
(Little Endian).
// For this reason the data bytes are swapped before being saved.


// Find the Maximum amplitude in I and Q to normalize the data
between +-1
maxAmp = Iwave[0];
minAmp = Iwave[0];
for( i=0; i<points; i++)
{
  if( maxAmp < Iwave[i] )
  maxAmp = Iwave[i];
   else if( minAmp > Iwave[i] )
      minAmp = Iwave[i];

  if( maxAmp < Qwave[i] )
```

```
        maxAmp = Qwave[i];
      else if( minAmp > Qwave[i] )
        minAmp = Qwave[i];
    }
  maxAmp = fabs(maxAmp);
  minAmp = fabs(minAmp);
  if( minAmp > maxAmp )
    maxAmp = minAmp;


  // Convert to short integers and interleave I/Q data
  scale = 32767 / maxAmp;      // Watch out for divide by zero.
  for( i=0; i<points; i++)
  {
    waveform[2*i] = (short)floor(Iwave[i]*scale + 0.5);
    waveform[2*i+1] = (short)floor(Qwave[i]*scale + 0.5);
  }
  // If on a PC swap the bytes to Big Endian
  if( strcmp(computer,"PCWIN") == 0 )
//if( PC )
  {
      pChar = (char *)&waveform[0];   // Character pointer to short
int data
      for( i=0; i<2*points; i++ )
      {
        buf = *pChar;
        *pChar = *(pChar+1);
        *(pChar+1) = buf;
        pChar+= 2;
      }
    }
  // Save the data to a file
  // Use FTP or one of the download assistants to download the file
to the
  // signal generator
      char *filename = "C:\\Temp\\PSGTestFile";
```

```
        FILE *stream = NULL;

        stream = fopen(filename, "w+b");// Open the file

    if (stream==NULL) perror ("Cannot Open File");

        int numwritten = fwrite( (void *)waveform, sizeof( short ),
points*2, stream );

    fclose(stream);// Close the file


        // 3.) Load the internal Arb format file
********************************
        // This process is just the reverse of saving the waveform

        // Read in waveform as unsigned short integers.

        // Swap the bytes as necessary

        // Normalize between +-1

        // De-interleave the I/Q Data

        // Open the file and load the internal format data

        stream = fopen(filename, "r+b");// Open the file

        if (stream==NULL) perror ("Cannot Open File");

        int numread = fread( (void *)waveform, sizeof( short ),
points*2, stream );

    fclose(stream);// Close the file

      // If on a PC swap the bytes back to Little Endian

        if( strcmp(computer,"PCWIN") == 0 )

    {

      pChar = (char *)&waveform[0];    // Character pointer to short
int data

        for( i=0; i<2*points; i++ )

        {

            buf = *pChar;

            *pChar = *(pChar+1);

            *(pChar+1) = buf;

            pChar+= 2;

        }
        }

        // Normalize De-Interleave the IQ data

        double Iwaveln[POINTS];

        double Qwaveln[POINTS];
```

```
for( i=0; i<points; i++)
{
   IwaveIn[i] = waveform[2*i] / 32767.0;
   QwaveIn[i] = waveform[2*i+1] / 32767.0;
}
return 0;
}
```

## Creating and Storing I/Q Data—Little Endian Order

On the documentation CD, this programming example's name is
"*CreateStore_Data_c++.txt.*"

This C++ programming example (compiled using Metrowerks CodeWarrior 3.0)
performs the following functions:

— error checking
— data creation
— byte swapping and interleaving for little endian order data
— binary data file storing to a PC or workstation

After creating the binary file, you can use FTP, one of the download utilities, or
one of the C++ download programming examples to download the file to the
signal generator.

```cpp
#include <iostream>

#include <fstream>

#include <math.h>

#include <stdlib.h>


using namespace std;


int main ( void )

{

    ofstream out_stream;         // write the I/Q data to a file

    const unsigned int SAMPLES =200;     // number of sample pairs in
the waveform

    const short AMPLITUDE = 32000;        // amplitude between 0 and
full scale dac value

    const double two_pi = 6.2831853;


    //allocate buffer for waveform

    short* iqData = new short[2*SAMPLES]; // need two bytes for each
integer

    if (!iqData)

    {

       cout << "Could not allocate data buffer." << endl;

       return 1;

    }
```

```
    out_stream.open("IQ_data");// create a data file

    if (out_stream.fail())

    {

      cout << "Input file opening failed" << endl;

      exit(1);

    }

    //generate the sample data for I and Q. The I channel will have
a sine

    //wave and the Q channel will a cosine wave.


    for (int i=0; i<SAMPLES; ++i)

    {

        iqData[2*i] = AMPLITUDE * sin(two_pi*i/(float)SAMPLES);

        iqData[2*i+1] = AMPLITUDE * cos(two_pi*i/(float)SAMPLES);

    }

   // make sure bytes are in the order MSB(most significant byte)
first. (PC only).


    char* cptr = (char*)iqData;// cast the integer values to
characters


    for (int i=0; i<(4*SAMPLES); i+=2)// 4*SAMPLES

    {

        char temp = cptr[i];// swap LSB and MSB bytes

        cptr[i]=cptr[i+1];

        cptr[i+1]=temp;

    }


    // now write the buffer to a file


        out_stream.write((char*)iqData, 4*SAMPLES);

 return 0;

}
```

## Creating and Downloading I/Q Data—Big and Little Endian Order

On the documentation CD, this programming example's name is
*"CreateDwnLd_Data_c++.txt."*

This C++ programming example (compiled using Microsoft Visual C++ 6.0)
performs the following functions:

— error checking
— data creation
— data scaling
— text file creation for viewing and debugging data
— byte swapping and interleaving for little endian order data
— interleaving for big endian order data
— data saving to an array (data block)
— data block download to the signal generator

```
// This C++ program is an example of creating and scaling

// I and Q data, and then downloading the data into the

// signal generator as an interleaved I/Q file.

// This example uses a sine and cosine wave as the I/Q

// data.

//

// Include the standard headers for SICL programming

#include <sicl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <math.h>


// Choose a GPIB, LAN, or RS-232 connection

char* instOpenString ="lan[galqaDhcp1]";

//char* instOpenString ="gpib0,19";


// Pick some maximum number of samples, based on the

// amount of memory in your computer and the signal generator.

const int NUMSAMPLES=500;


int main(int argc, char* argv[])
```

```
{
    // Create a text file to view the waveform
    // prior to downloading it to the signal generator.
    // This verifies that the data looks correct.

    char *ofile = "c:\\temp\\iq.txt";

    // Create arrays to hold the I and Q data

    int idata[NUMSAMPLES];
    int qdata[NUMSAMPLES];

    // save the number of sampes into numsamples
    int numsamples = NUMSAMPLES;

    // Fill the I and Q buffers with the sample data
    for(int index=0; index<numsamples; index++)
    {
        // Create the I and Q data for the number of waveform
        // points and Scale the data (20000 * ...) as a precentage
        // of the DAC full scale (-32768 to 32767). This example
        // scales to approximately 70% of full scale.
        idata[index]=23000 * sin((4*3.14*index)/numsamples);
        qdata[index]=23000 * cos((4*3.14*index)/numsamples);
    }

    // Print the I and Q values to a text file. View the data
    // to see if its correct and if needed, plot the data in a
    // spreadsheet to help spot any problems.
    FILE *outfile = fopen(ofile, "w");

    if (outfile==NULL) perror ("Error opening file to write");
    for(index=0; index<numsamples; index++)
    {
```

```
        fprintf(outfile, "%d, %d\n", idata[index], qdata[index]);
    }
    fclose(outfile);


    // Little endian order data, use the character array and for
loop.
    // If big endian order, comment out this character array and for
loop,
    // and use the next loop (Big Endian order data).

    // We need a buffer to interleave the I and Q data.
    // 4 bytes to account for 2 I bytes and 2 Q bytes.

    char iqbuffer[NUMSAMPLES*4];

    // Interleave I and Q, and swap bytes from little
    // endian order to big endian order.
    for(index=0; index<numsamples; index++)
    {
        int ivalue = idata[index];
        int qvalue = qdata[index];
        iqbuffer[index*4]   = (ivalue >> 8) & 0xFF; // high byte of i
        iqbuffer[index*4+1] = ivalue & 0xFF;        // low byte of i
        iqbuffer[index*4+2] = (qvalue >> 8) & 0xFF; // high byte of q
        iqbuffer[index*4+3] = qvalue & 0xFF;        // low byte of q
    }


    // Big Endian order data, uncomment the following lines of code.
    // Interleave the I and Q data.


    // short iqbuffer[NUMSAMPLES*2];            // Big endian order,
uncomment this line
    // for(index=0; index<numsamples; index++)  // Big endian order,
uncomment this line
    // {                                         // Big endian order,
uncomment this line
```

```
//     iqbuffer[index*2]   = idata[index];  // Big endian order,
uncomment this line

//     iqbuffer[index*2+1] = qdata[index];  // Big endian order,
uncomment this line

// }                                         // Big endian order,
uncomment this line


// Open a connection to write to the instrument

INST id=iopen(instOpenString);

if (!id)

{

    fprintf(stderr, "iopen failed (%s)\n", instOpenString);

    return -1;

}


// Declare variables to hold portions of the SCPI command

int bytesToSend;

char s[20];

char cmd[200];


bytesToSend = numsamples*4;        // calculate the number of
bytes

sprintf(s, "%d", bytesToSend); // create a string s with that
number of bytes


// The SCPI command has four parts.

//   Part 1 = :MEM:DATA "filename",#

//   Part 2 = length of Part 3 when written to a string

//   Part 3 = length of the data in bytes.  This is in s from
above.

//   Part 4 = the buffer of data


// Build parts 1, 2, and 3 for the I and Q data.

sprintf(cmd, ":MEM:DATA \"WFM1:FILE1\", #%d%d", strlen(s),
bytesToSend);

// Send parts 1, 2, and 3
```

```
    iwrite(id, cmd, strlen(cmd), 0, 0);

    // Send part 4.  Be careful to use the correct command here.  In
many

    // programming languages, there are two methods to send SCPI
commands:

    //   Method 1 = stop at the first '0' in the data

    //   Method 2 = send a fixed number of bytes, ignoring '0' in the
data.

    // You must find and use the correct command for Method 2.

    iwrite(id, iqbuffer, bytesToSend, 0, 0);

    // Send a terminating carriage return

    iwrite(id, "\n", 1, 1, 0);


    printf("Loaded file using the E4438C, E8267C and E8267D
format\n");

    return 0;
}
```

## Importing and Downloading I/Q Data—Big Endian Order

On the documentation CD, this programming example's name is
"*impDwnLd_c++.txt.*"

This C++ programming example (compiled using Metrowerks CodeWarrier 3.0)
assumes that the data is in big endian order and performs the following
functions:

— error checking
— binary file importing from the PC or workstation.
— binary file download to the signal generator.

```
// Description: Send a file in blocks of data to a signal generator

//

#include <sicl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>


// ATTENTION:

// - Configure these three lines appropriately for your instrument

//   and use before compiling and running
//

char* instOpenString = "gpib7,19"; //for LAN replace with
"lan[<hostname or IP address>]"

const char* localSrcFile = "D:\\home\\TEST_WAVE"; //enter file
location on PC/workstation

const char* instDestFile = "/USER/BBG1/WAVEFORM/TEST_WAVE"; //for
non-volatile memory

  //remove BBG1 from file path

// Size of the copy buffer

const int BUFFER_SIZE = 100*1024;


int

main()

{

    INST id=iopen(instOpenString);

    if (!id)
```

```
{
    fprintf(stderr, "iopen failed (%s)\n", instOpenString);
    return -1;
}

FILE* file = fopen(localSrcFile, "rb");
if (!file)
{
    fprintf(stderr, "Could not open file: %s\n", localSrcFile);
    return 0;
}

if( fseek( file, 0, SEEK_END ) < 0 )
{
    fprintf(stderr,"Cannot seek to the end of file.\n" );
    return 0;
}

long lenToSend = ftell(file);
printf("File size = %d\n", lenToSend);

if (fseek(file, 0, SEEK_SET) < 0)
{
    fprintf(stderr,"Cannot seek to the start of file.\n");
    return 0;
}

char* buf = new char[BUFFER_SIZE];
if (buf && lenToSend)
{
    // Prepare and send the SCPI command header
    char s[20];
    sprintf(s, "%d", lenToSend);
    int lenLen = strlen(s);
```

```
        char s2[256];

        sprintf(s2, "mmem:data \"%s\", #%d%d", instDestFile, lenLen,
lenToSend);

        iwrite(id, s2, strlen(s2), 0, 0);


        // Send file in BUFFER_SIZE chunks

        long numRead;

        do

        {

            numRead = fread(buf, sizeof(char), BUFFER_SIZE, file);

            iwrite(id, buf, numRead, 0, 0);

        } while (numRead == BUFFER_SIZE);


        // Send the terminating newline and EOM

        iwrite(id, "\n", 1, 1, 0);


        delete [] buf;

    }

    else

    {

        fprintf(stderr, "Could not allocate memory for copy
buffer\n");

    }


    fclose(file);

    iclose(id);

    return 0;

}
```

## Importing and Downloading Using VISA—Big Endian Order

On the documentation CD, this programming example's name is
*"DownLoad_Visa_c++.txt."*

This C++ programming example (compiled using Microsoft Visual C++ 6.0)
assumes that the data is in big endian order and performs the following
functions:

— error checking

— binary file importing from the PC or workstation

— binary file download to the signal generator's non–volatile memory

To load the waveform data to volatile (WFM1) memory, change the instDestfile declaration to: "USER/BBG1/WAVEFORM/".

```
//*****************************************************************

// PROGRAM NAME:Download_Visa_c++.cpp

//

// PROGRAM DESCRIPTION:Sample test program to download ARB waveform data. Send a

// file in chunks of ascii data to the signal generator.

//

// NOTE: You must have the IO Libraries installed to run this program.

//

// This example uses the LAN/TCPIP to download a file to the signal generator's

// non-volatile memory. The program allocates a memory buffer on the PC or

// workstation of 102400 bytes (100*1024 bytes). The actual size of the buffer is

// limited by the memory on your PC or workstation, so the buffer size can be

// increased or decreased to meet your system limitations.

//

// While this program uses the LAN/TCPIP to download a waveform file into

// non-volatile memory, it can be modified to store files in volatile memory

// WFM1 using GPIB by setting the instrOpenString = "TCPIP0::xxx.xxx.xxx.xxx::INSTR"

// declaration with "GPIB::19::INSTR"

//

// The program also includes some error checking to alert you when problems arise

// while trying to download files. This includes checking to see if the file exists.

//*****************************************************************

// IMPORTANT: Replace the xxx.xxx.xxx.xxx IP address in the instOpenString declaration
```

```
// in the code below with the IP address of your signal generator.
(or you can use the

// instrument's hostname). Replace the localSrcFile and instDestFile
directory paths

// as needed.
//*************************************************************



#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include  "visa.h"

//

// IMPORTANT:

//   Configure the following three lines correctly before compiling
and running


char* instOpenString ="TCPIP0::xxx.xxx.xxx.xxx::INSTR";  // your
instrument's IP address


const char* localSrcFile = "\\Files\\IQ_DataC";


const char* instDestFile = "/USER/WAVEFORM/IQ_DataC";


const int BUFFER_SIZE = 100*1024;// Size of the copy buffer


int main(int argc, char* argv[])

{

    ViSession defaultRM, vi;

    ViStatus status = 0;


    status = viOpenDefaultRM(&defaultRM);// Open the default
resource manager


    // TO DO: Error handling here
```

```
    status = viOpen(defaultRM, instOpenString, VI_NULL, VI_NULL,
&vi);


    if (status)// If any errors then display the error and exit the
program
    {
        fprintf(stderr, "viOpen failed (%s)\n", instOpenString);
    return -1;
    }


    FILE* file = fopen(localSrcFile, "rb");// Open local source file
for binary reading


    if (!file) // If any errors display the error and exit the
program
    {
        fprintf(stderr, "Could not open file: %s\n", localSrcFile);
    return 0;
    }


    if( fseek( file, 0, SEEK_END ) < 0 )
    {
        fprintf(stderr,"Cannot lseek to the end of file.\n" );
        return 0;
    }


    long lenToSend = ftell(file);// Number of bytes in the file


    printf("File size = %d\n", lenToSend);


    if (fseek(file, 0, SEEK_SET) < 0)
    {
        fprintf(stderr,"Cannot lseek to the start of file.\n");
        return 0;
    }
```

```cpp
    unsigned char* buf = new unsigned char[BUFFER_SIZE]; // Allocate
char buffer memory


    if (buf && lenToSend)

    {

        // Do not send the EOI (end of instruction) terminator on any
write except the

        // last one


        viSetAttribute( vi, VI_ATTR_SEND_END_EN, 0 );


        // Prepare and send the SCPI command header


        char s[20];
        sprintf(s, "%d", lenToSend);


        int lenLen = strlen(s);
        unsigned char s2[256];


    // Write the command mmem:data and the header. The number lenLen
represents the
    // number of bytes and the actual number of bytes is the variable
lenToSend


        sprintf((char*)s2, "mmem:data \"%s\", #%d%d", instDestFile,
lenLen, lenToSend);


    // Send the command and header to the signal generator


        viWrite(vi, s2, strlen((char*)s2), 0);


        long numRead;


    // Send file in BUFFER_SIZE chunks to the signal generator
```

```
        do

        {

            numRead = fread(buf, sizeof(char), BUFFER_SIZE, file);


            viWrite(vi, buf, numRead, 0);


        } while (numRead == BUFFER_SIZE);


        // Send the terminating newline and EOI


        viSetAttribute( vi, VI_ATTR_SEND_END_EN, 1 );


        char* newLine = "\n";


        viWrite(vi, (unsigned char*)newLine, 1, 0);


        delete [] buf;

    }

    else

    {

        fprintf(stderr, "Could not allocate memory for copy
buffer\n");

    }


    fclose(file);

    viClose(vi);

    viClose(defaultRM);


    return 0;

}
```

## Importing, Byte Swapping, Interleaving, and Downloading I and Q Data—Big and Little Endian Order

On the documentation CD, this programming example's name is
"*impDwnLd2_c++.txt.*"

This C++ programming example (compiled using Microsoft Visual C++ 6.0) performs the following functions:

— error checking
— binary file importing (earlier E443xB or current model signal generators)
— byte swapping and interleaving for little endian order data
— data interleaving for big endian order data
— data scaling
— binary file download for earlier E443xB data or current signal generator formatted data

```cpp
// This C++ program is an example of loading I and Q

// data into an E443xB, E4438C, E8267C, or E8267D signal

// generator.

//

// It reads the I and Q data from a binary data file

// and then writes the data to the instrument.


// Include the standard headers for SICL programming

#include <sicl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>


// Choose a GPIB, LAN, or RS-232 connection

char* instOpenString ="gpib0,19";


// Pick some maximum number of samples, based on the

// amount of memory in your computer and your waveforms.

const int MAXSAMPLES=50000;


int main(int argc, char* argv[])
```

```
{
    // These are the I and Q input files.
    // Some compilers will allow '/' in the directory
    // names.  Older compilers might need '\\' in the
    // directory names.  It depends on your operating system
    // and compiler.
    char *ifile = "c:\\SignalGenerator\\data\\BurstA1I.bin";
    char *qfile = "c:\\SignalGenerator\\data\\BurstA1Q.bin";

    // This is a text file to which we will write the
    // I and Q data just for debugging purposes.  It is
    // a good programming practice to check your data
    // in this way before attempting to write it to
    // the instrument.
    char *ofile = "c:\\SignalGenerator\\data\\iq.txt";

    // Create arrays to hold the I and Q data
    int idata[MAXSAMPLES];
    int qdata[MAXSAMPLES];

    // Often we must modify, scale, or offset the data
    // before loading it into the instrument.  These
    // buffers are used for that purpose.  Since each
    // sample is 16 bits, and a character only holds
    // 8 bits, we must make these arrays twice as long
    // as the I and Q data arrays.
    char ibuffer[MAXSAMPLES*2];
    char qbuffer[MAXSAMPLES*2];

    // For the E4438C or E8267C/67D, we might also need to interleave
    // the I and Q data.  This buffer is used for that
    // purpose.  In this case, this buffer must hold
    // both I and Q data so it needs to be four times
    // as big as the data arrays.
```

```
char iqbuffer[MAXSAMPLES*4];


// Declare variables which will be used later
bool done;
FILE *infile;
int index, numsamples, i1, i2, ivalue;


// In this example, we'll assume the data files have
// the I and Q data in binary form as unsigned 16 bit integers.
// This next block reads those binary files.  If your
// data is in some other format, then replace this block
// with appropriate code for reading your format.
// First read I values
done = false;
index = 0;
infile = fopen(ifile, "rb");
if (infile==NULL) perror ("Error opening file to read");
while(!done)
{
    i1 = fgetc(infile);   // read the first byte
    if(i1==EOF) break;
    i2 = fgetc(infile);   // read the next byte
    if(i2==EOF) break;
    ivalue=i1+i2*256;     // put the two bytes together
    // note that the above format is for a little endian
    // processor such as Intel.  Reverse the order for
    // a big endian processor such as Motorola, HP, or Sun
    idata[index++]=ivalue;
    if(index==MAXSAMPLES) break;
}
fclose(infile);


// Then read Q values
index = 0;
```

```
infile = fopen(qfile, "rb");

if (infile==NULL) perror ("Error opening file to read");

while(!done)

{

    i1 = fgetc(infile);   // read the first byte

    if(i1==EOF) break;

    i2 = fgetc(infile);   // read the next byte

    if(i2==EOF) break;

    ivalue=i1+i2*256;     // put the two bytes together

    // note that the above format is for a little endian

    // processor such as Intel.  Reverse the order for

    // a big endian processor such as Motorola, HP, or Sun

    qdata[index++]=ivalue;

    if(index==MAXSAMPLES) break;

}

fclose(infile);


// Remember the number of samples which were read from the file.

numsamples = index;


// Print the I and Q values to a text file.  If you are

// having trouble, look in the file and see if your I and

// Q data looks correct.  Plot the data from this file if

// that helps you to diagnose the problem.

FILE *outfile = fopen(ofile, "w");

if (outfile==NULL) perror ("Error opening file to write");

for(index=0; index<numsamples; index++)

{

    fprintf(outfile, "%d, %d\n", idata[index], qdata[index]);

}

fclose(outfile);


// The E443xB, E4438C, E8267C or E8267D all use big-endian

// processors.  If your software is running on a little-endian
```

```
// processor such as Intel, then you will need to swap the
// bytes in the data before sending it to the signal generator.


// The arrays ibuffer and qbuffer are used to hold the data
// after any byte swapping, shifting or scaling.


// In this example, we'll assume that the data is in the format
// of the E443xB without markers.  In other words, the data
// is in the range 0-16383.
//  0 gives negative full-scale output
//  8192 gives 0 V output
//  16383 gives positive full-scale output
// If this is not the scaling of your data, then you will need
// to scale your data appropriately in the next two blocks.


// ibuffer and qbuffer will hold the data in the E443xB format.
// No scaling is needed, however we need to swap the byte order
// on a little endian computer.  Remove the byte swapping
// if you are using a big endian computer.
for(index=0; index<numsamples; index++)
{
    int ivalue = idata[index];
    int qvalue = qdata[index];
    ibuffer[index*2]   = (ivalue >> 8) & 0xFF; // high byte of i
    ibuffer[index*2+1] = ivalue & 0xFF;         // low byte of i
    qbuffer[index*2]   = (qvalue >> 8) & 0xFF; // high byte of q
    qbuffer[index*2+1] = qvalue & 0xFF;         // low byte of q
}


// iqbuffer will hold the data in the E4438C, E8267C, E8267D
// format.  In this format, the I and Q data is interleaved.
// The data is in the range -32768 to 32767.
//   -32768 gives negative full-scale output
//         0 gives 0 V output
```

```
//     32767 gives positive full-scale output
// From these ranges, it appears you should offset the
// data by 8192 and scale it by 4.  However, due to the
// interpolators in these products, it is better to scale
// the data by a number less than four.  Commonly a good
// choice is 70% of 4 which is 2.8.
// By default, the signal generator scales data to 70%
// If you scale the data here, you may want to change the
// signal generator scaling to 100%
// Also we need to swap the byte order on a little endian
// computer.  This code also works for big endian order data
// since it swaps bytes based on the order.
for(index=0; index<numsamples; index++)
{
    int iscaled = 2.8*(idata[index]-8192); // shift and scale
    int qscaled = 2.8*(qdata[index]-8192); // shift and scale
    iqbuffer[index*4]   = (iscaled >> 8) & 0xFF; // high byte of i
    iqbuffer[index*4+1] = iscaled & 0xFF;        // low byte of i
    iqbuffer[index*4+2] = (qscaled >> 8) & 0xFF; // high byte of q
    iqbuffer[index*4+3] = qscaled & 0xFF;   // low byte of q
}

// Open a connection to write to the instrument
INST id=iopen(instOpenString);
if (!id)
{
    fprintf(stderr, "iopen failed (%s)\n", instOpenString);
    return -1;
}

// Declare variables which will be used later
int bytesToSend;
```

```
    char s[20];

    char cmd[200];


    // The E4438C, E8267C and E8267D accept the E443xB format.

    // so we can use this next section on any of these signal
generators.

    // However the E443xB format only uses 14 bits.


    bytesToSend = numsamples*2;         // calculate the number of
bytes

    sprintf(s, "%d", bytesToSend); // create a string s with that
number of bytes


    // The SCPI command has four parts.

    //   Part 1 = :MEM:DATA "filename",

    //   Part 2 = length of Part 3 when written to a string

    //   Part 3 = length of the data in bytes.  This is in s from
above.

    //   Part 4 = the buffer of data


    // Build parts 1, 2, and 3 for the I data.

    sprintf(cmd, ":MEM:DATA \"ARBI:FILE1\", #%d%d", strlen(s),
bytesToSend);

    // Send parts 1, 2, and 3

    iwrite(id, cmd, strlen(cmd), 0, 0);

    // Send part 4.  Be careful to use the correct command here.  In
many

    // programming languages, there are two methods to send SCPI
commands:

    //   Method 1 = stop at the first '0' in the data

    //   Method 2 = send a fixed number of bytes, ignoring '0' in the
data.

    // You must find and use the correct command for Method 2.

    iwrite(id, ibuffer, bytesToSend, 0, 0);

    // Send a terminating carriage return

    iwrite(id, "\n", 1, 1, 0);
```

```
    // Identical to the section above, except for the Q data.

    sprintf(cmd, ":MEM:DATA \"ARBQ:FILE1\", #%d%d",
strlen(s),bytesToSend);

    iwrite(id, cmd, strlen(cmd), 0, 0);

    iwrite(id, qbuffer, bytesToSend, 0, 0);

    iwrite(id, "\n", 1, 1, 0);


    printf("Loaded FILE1 using the E443xB format\n");


    // The E4438C, E8267C and E8267D have a newer faster format which

    // allows 16 bits to be used.  However this format is not
accepted in

    // the E443xB.  Therefore do not use this next section for the
E443xB.


    printf("Note: Loading FILE2 on a E443xB will cause \"ERROR: 208,
I/O error\"\n");


    // Identical to the I and Q sections above except

    //   a) The I and Q data are interleaved

    //   b) The buffer of I+Q is twice as long as the I buffer was.

    //   c) The SCPI command uses WFM1 instead of ARBI and ARBQ.

    bytesToSend = numsamples*4;

    sprintf(s, "%d", bytesToSend);

        sprintf(cmd, ":mem:data \"WFM1:FILE2\", #%d%d",
strlen(s),bytesToSend);

    iwrite(id, cmd, strlen(cmd), 0, 0);

    iwrite(id, iqbuffer, bytesToSend, 0, 0);

    iwrite(id, "\n", 1, 1, 0);

    printf("Loaded FILE2 using the E4438C, E8267C and E8267D
format\n");

    return 0;

}
```

## MATLAB Programming Examples

This section contains the following programming examples:

### Creating and Storing I/Q Data

On the documentation CD, this programming example's name is
"*offset_iq_ml.m.*"

This MATLAB programming example follows the same coding algorithm as the
C++ programming example "Creating and Storing Offset I/Q Data—Big and
Little Endian Order" on page  58 and performs the following functions:

— error checking
— data creation
— data normalization
— data scaling
— I/Q signal offset from the carrier (single sideband suppressed carrier signal)
— byte swapping and interleaving for little endian order data
— I and Q interleaving for big endian order data
— binary data file storing to a PC or workstation
— reversal of the data formatting process (byte swapping, interleaving, and
  normalizing the data)

```
function main

% Using MatLab this example shows how to

% 1.) Create a simple IQ waveform

% 2.) Save the waveform into the Keysight MXG/ESG/PSG Internal Arb
format

%      This format is for the N5162A/82A, E4438C, E8267C, and E8267D

%      This format will not work with the earlier E443xB ESG

% 3.) Load the internal Arb format file into a MatLab array


% 1.) Create Simple IQ Signal
*****************************************

% This signal is a single tone on the upper

% side of the carrier and is usually refered to as

% a Single Side Band Suppressed Carrier (SSBSC) signal.

% It is nothing more than a cosine wavefomm in I

% and a sine waveform in Q.

%
```

```
points = 1000;        % Number of points in the waveform

cycles = 101;         % Determines the frequency offset from the
carrier


phaseInc = 2*pi*cycles/points;

phase = phaseInc * [0:points-1];


Iwave = cos(phase);

Qwave = sin(phase);


% 2.) Save waveform in internal format
********************************
% Convert the I and Q data into the internal arb format

% The internal arb format is a single waveform containing
interleaved IQ

% data. The I/Q data is signed short integers (16 bits).

% The data has values scaled between +-32767 where

%   DAC Value    Description

%    32767       Maximum positive value of the DAC

%        0       Zero out of the DAC

%    -32767      Maximum negative value of the DAC

% The internal arb expects the data bytes to be in Big Endian
format.

% This is opposite of how short integers are saved on a PC (Little
Endian).

% For this reason the data bytes are swapped before being saved.


% Interleave the IQ data

waveform(1:2:2*points) = Iwave;

waveform(2:2:2*points) = Qwave;

%[Iwave;Qwave];

%waveform = waveform(:)';


% Normalize the data between +-1

waveform = waveform / max(abs(waveform));    % Watch out for divide
by zero.
```

```
% Scale to use full range of the DAC

waveform = round(waveform * 32767);      % Data is now effectively
signed short integer values


% waveform = round(waveform * (32767 / max(abs(waveform))));     %
More efficient than previous two steps!


% PRESERVE THE BIT PATTERN but convert the waveform to

% unsigned short integers so the bytes can be swapped.

% Note: Can't swap the bytes of signed short integers in MatLab.

waveform = uint16(mod(65536 + waveform,65536)); %


% If on a PC swap the bytes to Big Endian

if strcmp( computer, 'PCWIN' )

    waveform = bitor(bitshift(waveform,-8),bitshift(waveform,8));

end


% Save the data to a file

% Note: The waveform is saved as unsigned short integers. However,

%       the acual bit pattern is that of signed short integers and

%       that is how the Keysight MXG/ESG/PSG interprets them.

filename = 'C:\Temp\PSGTestFile';

[FID, message] = fopen(filename,'w');% Open a file to write data

if FID == -1 error('Cannot Open File'); end

fwrite(FID,waveform,'unsigned short');% write to the file

fclose(FID);                            % close the file


% 3.) Load the internal Arb format file
*******************************

% This process is just the reverse of saving the waveform

% Read in waveform as unsigned short integers.

% Swap the bytes as necessary

% Convert to signed integers then normalize between +-1

% De-interleave the I/Q Data
```

```
% Open the file and load the internal format data

[FID, message] = fopen(filename,'r');% Open file to read data

if FID == -1 error('Cannot Open File'); end

[internalWave,n] = fread(FID, 'uint16');% read the IQ file

fclose(FID);% close the file


internalWave = internalWave';    % Conver from column array to row
array


% If on a PC swap the bytes back to Little Endian

if strcmp( computer, 'PCWIN' )  % Put the bytes into the correct
order

    internalWave=
bitor(bitshift(internalWave,-8),bitshift(bitand(internalWave,255),8
));

end


% convert unsigned to signed representation

internalWave = double(internalWave);

tmp = (internalWave > 32767.0) * 65536;

iqWave = (internalWave - tmp) ./ 32767;   % and normalize the data


% De-Interleave the IQ data

IwaveIn = iqWave(1:2:n);

QwaveIn = iqWave(2:2:n);
```

## Creating and Downloading a Pulse

**NOTE**    This section applies only to the Keysight MXG and the PSG.

For the Keysight MXG, the maximum frequency is 6 GHz, and the **pulsepat.m** program's `SOURce:FREQuency 20000000000` value must be changed as required in the following programs. For more frequency information, refer to the signal generator's **Data Sheet**.

On the documentation CD, this programming example's name is "*pulsepat.m*."

This MATLAB programming example performs the following functions:

— I and Q data creation for 10 pulses
— marker file creation
— data scaling
— downloading using Waveform Download Assistant functions (see for more information)

```
% Script file: pulsepat.m

%

% Purpose:

%   To calculate and download an arbitrary waveform file that
simulates a

%   simple antenna scan pulse pattern to the Keysight MXG/PSG vector
signal generator.

%

% Define Variables:

%    n -- counting variable (no units)

%    t -- time (seconds)

%    rise -- raised cosine pulse rise-time definition (samples)

%    on -- pulse on-time definition (samples)

%    fall -- raised cosine pulse fall-time definition (samples)

%    i -- in-phase modulation signal

%    q -- quadrature modulation signal


n=4;                            % defines the number of points in the
rise-time and fall-time

t=-1:2/n:1-2/n;                 % number of points translated to time

rise=(1+sin(t*pi/2))/2;         % defines the pulse rise-time shape
```

```
on=ones(1,120);                 % defines the pulse on-time
characteristics

fall=(1+sin(-t*pi/2))/2;      % defines the pulse fall-time shape

off=zeros(1,896);           % defines the pulse off-time
characteristics


% arrange the i-samples and scale the amplitude to simulate an
antenna scan

% pattern comprised of 10 pulses

i = .707*[rise on fall off...

[.9*[rise on fall off]]...

    [.8*[rise on fall off]]...

    [.7*[rise on fall off]]...

    [.6*[rise on fall off]]...

    [.5*[rise on fall off]]...

    [.4*[rise on fall off]]...

    [.3*[rise on fall off]]...

    [.2*[rise on fall off]]...

    [.1*[rise on fall off]]];


% set the q-samples to all zeroes

q = zeros(1,10240);


% define a composite iq matrix for download to the Keysight MXG/PSG
using the

% Waveform Download Assistant

IQData = [i + (j * q)];


% define a marker matrix and activate a marker to indicate the
beginning of the waveform

Markers = zeros(2,length(IQData));      % fill marker array with
zero, i.e no markers set

Markers(1,1) = 1;        % set marker to first point of playback


% make a new connection to theKeysight MXG/PSG over the GPIB
interface
```

```
io = agt_newconnection('gpib',0,19);
```

```
% verify that communication with the Keysight MXG/PSG has been
established
[status, status_description, query_result] = agt_query(io,'*idn?');
if (status < 0) return; end
```

```
% set the carrier frequency and power level on the Keysight MXG/PSG
using the Keysight
%Waveform Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
20000000000');
[status, status_description] = agt_sendcommand(io, 'POWer 0');
```

```
% define the ARB sample clock for playback
sampclk = 40000000;
```

```
% download the iq waveform to the PSG baseband generator for
playback
[status, status_description] = agt_waveformload(io, IQData,
'pulsepat', sampclk, 'play', 'no_normscale', Markers);
```

```
% turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe
ON' )
```

You can test your program by performing a simulated plot of the in–phase modulation signal in Matlab (see ). To do this, enter `plot (i)` at the Matlab command prompt.

Figure 2                      Simulated Plot of In–Phase Signal
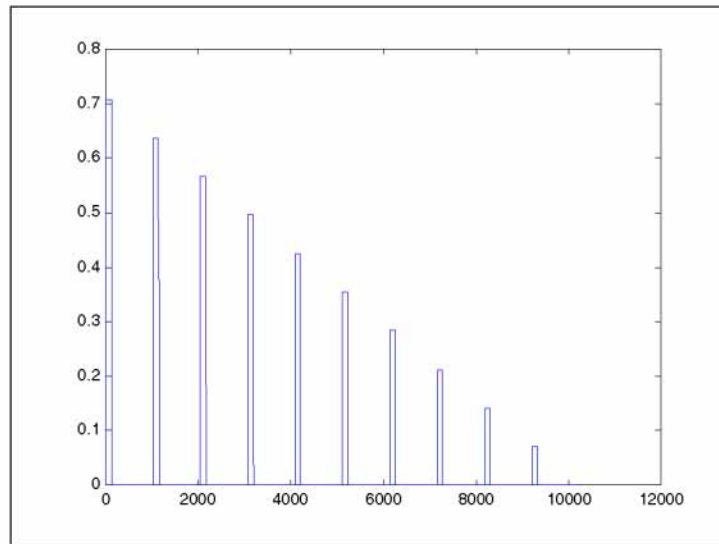


The following additional Matlab M–file pulse programming examples are also available on the
**Documentation CD–ROM** for your Keysight MXG and PSG signal generator:

**NOTE**                    For the Keysight MXG, the SOURce:FREQuency 20000000000 value must be changed as required in the following programs. For more information, refer to the **Data Sheet**.

---

| | |
|---|---|
| barker.m | This programming example calculates and downloads an arbitrary waveform file that simulates a simple 7–bit barker RADAR signal to the PSG vector signal generator. |
| chirp.m | This programming example calculates and downloads an arbitrary waveform file that simulates a simple compressed pulse RADAR signal using linear FM chirp to the PSG vector signal generator. |
| FM.m | This programming example calculates and downloads an arbitrary waveform file that simulates a single tone FM signal with a rate of 6 KHz, deviation of =/– 14.3 KHz, Bessel null of dev/rate=2.404 to the Keysight MXG/PSG vector signal generator. |
| nchirp.m | This programming example calculates and downloads an arbitrary waveform file that simulates a simple compressed pulse RADAR signal using non–linear FM chirp to the PSG vector signal generator. |

---

pulse.m

This programming example calculates and downloads an arbitrary waveform file that simulates a simple pulse signal to the PSG vector signal generator.

pulsedroop.m

This programming example calculates and downloads an arbitrary waveform file that simulates a simple pulse signal with pulse droop to the PSG vector signal generator.

## Visual Basic Programming Examples

### Creating I/Q Data—Little Endian Order

On the documentation CD, this programming example's name is
*"Create_IQData_vb.txt."*

This Visual Basic programming example, using Microsoft Visual Basic 6.0, uses
little endian order data, and performs the following functions:

— error checking
— I an Q integer array creation
— I an Q data interleaving
— byte swapping to convert to big endian order
— binary data file storing to a PC or workstation

Once the file is created, you can download the file to the signal generator using
FTP (see ).

```
'*****************************************************************

' Program Name: Create_IQData

' Program Description: This program creates a sine and cosine wave
using 200 I/Q data

' samples. Each I and Q value is represented by a 2 byte integer. The
sample points are

' calculated, scaled using the AMPLITUDE constant of 32767, and then
stored in an array

' named iq_data. The AMPLITUDE scaling allows for full range I/Q
modulator DAC values.

' Data must be in 2's complemant, MSB/LSB big-endian format. If your
PC uses LSB/MSB

' format, then the integer bytes must be swapped. This program
converts the integer

' array values to hex data types and then swaps the byte positions
before saving the

' data to the IQ_DataVB file.

'*****************************************************************

Private Sub Create_IQData()

Dim index As Integer

Dim AMPLITUDE As Integer

Dim pi As Double

Dim loByte As Byte

Dim hiByte As Byte
```

```
Dim loHex As String

Dim hiHex As String

Dim strSrc As String

Dim numPoints As Integer

Dim FileHandle As Integer

Dim data As Byte

Dim iq_data() As Byte

Dim strFilename As String


strFilename = "C:\IQ_DataVB"


Const SAMPLES = 200      ' Number of sample PAIRS of I and Q integers
for the waveform

AMPLITUDE = 32767        ' Scale the amplitude for full range of the
signal generators

                         ' I/Q modulator DAC

pi = 3.141592


Dim intIQ_Data(0 To 2 * SAMPLES - 1)  'Array for I and Q integers:
400

ReDim iq_data(0 To (4 * SAMPLES - 1)) 'Need MSB and LSB bytes for
each integer value:  800


'Create an integer array of I/Q pairs


    For index = 0 To (SAMPLES - 1)

        intIQ_Data(2 * index) = CInt(AMPLITUDE * Sin(2 * pi * index
/ SAMPLES))

        intIQ_Data(2 * index + 1) = CInt(AMPLITUDE * Cos(2 * pi *
index / SAMPLES))

    Next index


 'Convert each integer value to a hex string and then write into the
iq_data byte array

 'MSB, LSB ordered

 For index = 0 To (2 * SAMPLES - 1)
```

```
        strSrc = Hex(intIQ_Data(index)) 'convert the integer to a hex
value


    If Len(strSrc) <> 4 Then

        strSrc = String(4 - Len(strSrc), "0") & strSrc  'Convert to
hex format i.e "800F
    End If                                          'Pad with 0's
if needed to get 4

                                                     'characters
i.e '0' to "0000"


        hiHex = Mid$(strSrc, 1, 2)   'Get the first two hex values
(MSB)

        loHex = Mid$(strSrc, 3, 2)   'Get the next two hex values
(LSB)

        loByte = CByte("&H" & loHex) 'Convert to byte data type LSB

        hiByte = CByte("&H" & hiHex) 'Convert to byte data type MSB


        iq_data(2 * index) = hiByte     'MSB into first byte

        iq_data(2 * index + 1) = loByte  'LSB into second byte


  Next index


  'Now write the data to the file


FileHandle = FreeFile()      'Get a file number


numPoints = UBound(iq_data) 'Get the number of bytes in the file


Open strFilename For Binary Access Write As #FileHandle Len =
numPoints + 1


On Error GoTo file_error


    For index = 0 To (numPoints)
        data = iq_data(index)
```

```
        Put #FileHandle, index + 1, data 'Write the I/Q data to the
file

    Next index


Close #FileHandle


Call MsgBox("Data written to file " & strFilename, vbOKOnly,
"Download")


Exit Sub


file_error:

    MsgBox Err.Description

    Close #FileHandle


End Sub
```

## Downloading I/Q Data

On the signal generator's documentation CD, this programming example's name is "*Download_File_vb.txt.*"

This Visual Basic programming example, using Microsoft Visual Basic 6.0, downloads the file created in "Creating I/Q Data—Little Endian Order" on page 96 into non–volatile memory using a LAN connection. To use GPIB, replace the instOpenString object declaration with "GPIB::19::INSTR". To download the data into volatile memory, change the instDestfile declaration to "USER/BBG1/WAVEFORM/".

| | |
|---|---|
| **NOTE** | The example program listed here uses the VISA COM IO API, which includes the WriteIEEEBlock method. This method eliminates the need to format the download command with arbitrary block information such as defining number of bytes and byte numbers. Refer to "SCPI Command Line Structure" on page 29 for more information. |

This program also includes some error checking to alert you when problems arise while trying to download files. This includes checking to see if the file exists.

```
' ***************************************************************

' Program Name: Download_File
```

```
' Program Description: This program uses Microsoft Visual Basic 6.0
and the Keysight

' VISA COM I/O Library to download a waveform file to the signal
generator.

'

' The program downloads a file (the previously created 'IQ_DataVB'
file) to the signal

' generator. Refer to the Programming Guide for information on
binary

' data requirements for file downloads. The waveform data
'IQ_DataVB' is

' downloaded to the signal generator's non-volatile memory(NVWFM)

' " /USER/WAVEFORM/IQ_DataVB". For volatile memory(WFM1) download to
the

' " /USER/BBG1/WAVEFORM/IQ_DataVB" directory.

'

' You must reference the Keysight VISA COM Resource Manager and VISA
COM 1.0 Type

' Library in your Visual Basic project in the Project/References
menu.

' The VISA COM 1.0 Type Library, corresponds to VISACOM.tlb and the

' VISA COM Resource Manager, corresponds to AgtRM.DLL.

' The VISA COM 488.2 Formatted I/O 1.0, corresponds to the
BasicFormattedIO.dll

' Use a statement such as "Dim Instr As VisaComLib.FormattedIO488"
to

' create the formatted I/O reference and use

' "Set Instr = New VisaComLib.FormattedIO488" to create the actual
object.

' ****************************************************************

' IMPORTANT: Use the TCPIP address of your signal generator in the
rm.Open

' declaraion. If you are using the GPIB interface in your project
use "GPIB::19::INSTR"

' in the rm.Open declaration.

' ****************************************************************


Private Sub Download_File()
```

```
' The following four lines declare IO objects and instantiate them.

Dim rm As VisaComLib.ResourceManager

Set rm = New AgilentRMLib.SRMCls

Dim SigGen As VisaComLib.FormattedIO488

Set SigGen = New VisaComLib.FormattedIO488


' NOTE: Use the IP address of your signal generator in the rm.Open
declaration

Set SigGen.IO = rm.Open("TCPIP0::000.000.000.000")


Dim data As Byte

Dim iq_data() As Byte

Dim FileHandle As Integer

Dim numPoints As Integer

Dim index As Integer

Dim Header As String

Dim response As String

Dim hiByte As String

Dim loByte As String

Dim strFilename As String


strFilename = "C:\IQ_DataVB" 'File Name and location on PC

                                'Data will be saved to the signal
generator's NVWFM                           '/USER/WAVEFORM/IQ_DataVB
directory.


FileHandle = FreeFile()


On Error GoTo errorhandler


With SigGen               'Set up the signal generator to accept a
download
    .IO.Timeout = 5000     'Timeout 50 seconds
    .WriteString "*RST"    'Reset the signal generator.
End With
```

```
numPoints = (FileLen(strFilename))  'Get number of bytes in the
file: 800 bytes


ReDim iq_data(0 To numPoints - 1)      'Dimension the iq_data array
to the

                                       'size of the IQ_DataVB file:
800 bytes


Open strFilename For Binary Access Read As #FileHandle  'Open the
file for binary read
On Error GoTo file_error

For index = 0 To (numPoints - 1)    'Write the IQ_DataVB data to the
iq_data array
      Get #FileHandle, index + 1, data    '(index+1) is the record
number
        iq_data(index) = data
Next index


   Close #FileHandle                        'Close the file


'Write the command to the Header string. NOTE: syntax
     Header = "MEM:DATA ""/USER/WAVEFORM/IQ_DataVB"","


  'Now write the data to the signal generator's non-volatile memory
(NVWFM)


    SigGen.WriteIEEEBlock Header, iq_data


    SigGen.WriteString "*OPC?"               'Wait for the operation to
complete
    response = SigGen.ReadString         'Signal generator reponse to
the OPC? query
    Call MsgBox("Data downloaded to the signal generator", vbOKOnly,
"Download")
    Exit Sub
errorhandler:
```

```
      MsgBox Err.Description, vbExclamation, "Error Occurred",
Err.HelpFile, Err.HelpContext
```

```
Exit Sub
```

```
file_error:
```

```
      Call MsgBox(Err.Description, vbOKOnly)  'Display any error
message
```

```
      Close #FileHandle
```

```
End Sub
```

## HP Basic Programming Examples

This section contains the following programming examples:

### Creating and Downloading Waveform Data Using HP BASIC for Windows®

On the documentation CD, this programming example's name is "*hpbasicWin.txt.*"

The following program will download a waveform using HP Basic for Windows into volatile ARB memory. The waveform generated by this program is the same as the default `SINE_TEST_WFM` waveform file available in the signal generator's waveform memory. This code is similar to the code shown for BASIC for UNIX but there is a formatting difference in line 130 and line 140.

To download into non–volatile memory, replace line 190 with:

190 OUTPUT @PSG USING "#,K";":MMEM:DATA ""NVWFM:testfile"", #"

As discussed at the beginning of this section, I and Q waveform data is interleaved into one file in 2's compliment form and a marker file is associated with this I/Q waveform file.

In the `Output` commands, `USING "#,K"` formats the data. The pound symbol (#) suppresses the automatic EOL (End of Line) output. This allows multiple output commands to be concatenated as if they were a single output. The `"K"` instructs HP Basic to output the following numbers or strings in the default format.

```
10 !  RE-SAVE "BASIC_Win_file"
```

```
20      Num_points=200

30      ALLOCATE INTEGER Int_array(1:Num_points*2)

40      DEG

50      FOR I=1 TO Num_points*2 STEP 2

60        Int_array(I)=INT(32767*(SIN(I*360/Num_points)))

70      NEXT I

80      FOR I=2 TO Num_points*2 STEP 2

90        Int_array(I)=INT(32767*(COS(I*360/Num_points)))

100     NEXT I

110     PRINT "Data Generated"

120     Nbytes=4*Num_points

130     ASSIGN @PSG TO 719

140     ASSIGN @PSGb TO 719;FORMAT MSB FIRST

150     Nbytes$=VAL$(Nbytes)

160     Ndigits=LEN(Nbytes$)

170     Ndigits$=VAL$(Ndigits)

180     WAIT 1

190     OUTPUT @PSG USING "#,K";":MMEM:DATA ""WFM1:data_file"",#"

200     OUTPUT @PSG USING "#,K";Ndigits$

210     OUTPUT @PSG USING "#,K";Nbytes$

220     WAIT 1

230     OUTPUT @PSGb;Int_array(*)

240     OUTPUT @PSG;END

250     ASSIGN @PSG TO *

260     ASSIGN @PSGb TO *

270     PRINT

280     PRINT "*END*"

290     END
```

**Program Comments**

| | |
|---|---|
| 10: | Program file name |
| 20: | Sets the number of points in the waveform. |
| 30: | Allocates integer data array for I and Q waveform points. |
| 40: | Sets HP BASIC to use degrees for cosine and sine functions. |

**Program Comments  (Continued)**

| 50: | Sets up first loop for I waveform points. |
|---|---|
| 60: | Calculate and interleave I waveform points. |
| 70: | End of loop |
| 80: | Sets up second loop for Q waveform points. |
| 90: | Calculate and interleave Q waveform points. |
| 100: | End of loop. |
| 120: | Calculates number of bytes in I/Q waveform. |
| 130: | Opens an IO path to the signal generator using GPIB. 7 is the address of the GPIB card in the computer, and 19 is the address of the signal generator. This IO path is used to send ASCII data to the signal generator. |
| 140: | Opens an IO path for sending binary data to the signal generator. |
| 150: | Creates an ASCII string representation of the number of bytes in the waveform. |
| 160 to 170: | Finds the number of digits in Nbytes. |
| 190: | Sends the first part of the SCPI command, MEM:DATA along with the name of the file, `data_file`, that will receive the waveform data. The name, `data_file`, will appear in the signal generator's memory catalog. |
| 200 to 210: | Sends the rest of the ASCII header. |
| 230: | Sends the binary data. Note that `PSGb` is the binary IO path. |
| 240: | Sends an End–of–Line to terminate the transmission. |
| 250 to 260: | Closes the connections to the signal generator. |
| 290: | End the program. |

## Creating and Downloading Waveform Data Using HP BASIC for UNIX

On the documentation CD, this programming example's name is
"*hpbasicUx.txt.*"

The following program shows you how to download waveforms using HP Basic for UNIX. The code is similar to that shown for HP BASIC for Windows, but there is a formatting difference in line 130 and line 140.

To download into non–volatile memory, replace line 190 with:

190 OUTPUT @PSG USING "#,K";":MMEM:DATA ""NVWFM:testfile"", #"

As discussed at the beginning of this section, I and Q waveform data is interleaved into one file in 2's compliment form and a marker file is associated with this I/Q waveform file.

In the `Output` commands, `USING "#,K"` formats the data. The pound symbol (#) suppresses the automatic EOL (End of Line) output. This allows multiple output commands to be concatenated as if they were a single output. The "K" instructs HP BASIC to output the following numbers or strings in the default format.

```
10  !  RE-SAVE "UNIX_file"

20     Num_points=200

30     ALLOCATE INTEGER Int_array(1:Num_points*2)

40     DEG

50     FOR I=1 TO Num_points*2 STEP 2

60       Int_array(I)=INT(32767*(SIN(I*360/Num_points)))

70     NEXT I

80     FOR I=2 TO Num_points*2 STEP 2

90       Int_array(I)=INT(32767*(COS(I*360/Num_points)))

100    NEXT I

110    PRINT "Data generated "

120    Nbytes=4*Num_points

130    ASSIGN @PSG TO 719;FORMAT ON

140    ASSIGN @PSGb TO 719;FORMAT OFF

150    Nbytes$=VAL$(Nbytes)

160    Ndigits=LEN(Nbytes$)

170    Ndigits$=VAL$(Ndigits)

180    WAIT 1

190    OUTPUT @PSG USING "#,K";":MMEM:DATA ""WFM1:data_file""",#"

200    OUTPUT @PSG USING "#,K";Ndigits$

210    OUTPUT @PSG USING "#,K";Nbytes$

220    WAIT 1

230    OUTPUT @PSGb;Int_array(*)

240    WAIT 2

241    OUTPUT @PSG;END

250    ASSIGN @PSG TO *

260    ASSIGN @PSGb TO *

270    PRINT

280    PRINT "*END*"
```

290    END

**Program Comments**

| | |
|---|---|
| 10: | Program file name |
| 20: | Sets the number of points in the waveform. |
| 30: | Allocates integer data array for I and Q waveform points. |
| 40: | Sets HP BASIC to use degrees for cosine and sine functions. |
| 50: | Sets up first loop for I waveform points. |
| 60: | Calculate and interleave I waveform points. |
| 70: | End of loop |
| 80 | Sets up second loop for Q waveform points. |
| 90: | Calculate and interleave Q waveform points. |
| 100: | End of loop. |
| 120: | Calculates number of bytes in I/Q waveform. |
| 130: | Opens an IO path to the signal generator using GPIB. 7 is the address of the GPIB card in the computer, and 19 is the address of the signal generator. This IO path is used to send ASCII data to the signal generator. |
| 140: | Opens an IO path for sending binary data to the signal generator. |
| 150: | Creates an ASCII string representation of the number of bytes in the waveform. |
| 160 to 170: | Finds the number of digits in Nbytes. |
| 190: | Sends the first part of the SCPI command, MEM:DATA along with the name of the file, `data_file`, that will receive the waveform data. The name, `data_file,` will appear in the signal generator's memory catalog. |
| 200 to 210: | Sends the rest of the ASCII header. |
| 230: | Sends the binary data. Note that `PSGb` is the binary IO path. |
| 240: | Sends an End–of–Line to terminate the transmission. |
| 250 to 260: | Closes the connections to the signal generator. |
| 290: | End the program. |

### Creating and Downloading E443xB Waveform Data Using HP BASIC for Windows

On the documentation CD, this programming example's name is *"e443xb_hpbasicWin2.txt."*

The following program shows you how to download waveforms using HP Basic for Windows into volatile ARB memory. This program is similar to the following program example as well as the previous examples. The difference between BASIC for UNIX and BASIC for Windows is the way the formatting, for the most significant bit (MSB) on lines 110 and 120, is handled.

To download into non–volatile ARB memory, replace line 160 with:

160 OUTPUT @ESG USING "#,K";":MMEM:DATA ""NVARBI:testfile"", #"

and replace line 210 with:

210 OUTPUT @ESG USING "#,K";":MMEM:DATA ""NVARBQ:testfile"", #"

First, the I waveform data is put into an array of integers called `Iwfm_data` and the Q waveform data is put into an array of integers called Qwfm_data. The variable `Nbytes` is set to equal the number of bytes in the I waveform data. This should be twice the number of integers in `Iwfm_data`, since an integer is 2 bytes. Input integers must be between 0 and 16383.

In the `Output` commands, `USING "#,K"` formats the data. The pound symbol (#) suppresses the automatic EOL (End of Line) output. This allows multiple output commands to be concatenated as if they were a single output. The `"K"` instructs HP Basic to output the following numbers or strings in the default format.

```
10     ! RE-SAVE "ARB_IQ_Win_file"

20     Num_points=200

30     ALLOCATE INTEGER
Iwfm_data(1:Num_points),Qwfm_data(1:Num_points)

40     DEG

50     FOR I=1 TO Num_points

60       Iwfm_data(I)=INT(8191*(SIN(I*360/Num_points))+8192)

70       Qwfm_data(I)=INT(8191*(COS(I*360/Num_points))+8192)

80     NEXT I

90     PRINT "Data Generated"

100    Nbytes=2*Num_points

110    ASSIGN @Esg TO 719

120    !ASSIGN @Esgb TO 719;FORMAT MSB FIRST

130    Nbytes$=VAL$(Nbytes)

140    Ndigits=LEN(Nbytes$)

150    Ndigits$=VAL$(Ndigits)

160    OUTPUT @Esg USING "#,K";":MMEM:DATA ""ARBI:file_name_1"",#"

170    OUTPUT @Esg USING "#,K";Ndigits$

180    OUTPUT @Esg USING "#,K";Nbytes$

190    OUTPUT @Esgb;Iwfm_data(*)

200    OUTPUT @Esg;END

210    OUTPUT @Esg USING "#,K";":MMEM:DATA ""ARBQ:file_name_1"",#"
```

```
220    OUTPUT @Esg USING "#,K";Ndigits$

230    OUTPUT @Esg USING "#,K";Nbytes$

240    OUTPUT @Esgb;Qwfm_data(*)

250    OUTPUT @Esg;END

260    ASSIGN @Esg TO *

270    ASSIGN @Esgb TO *

280    PRINT

290    PRINT "*END*"

300    END
```
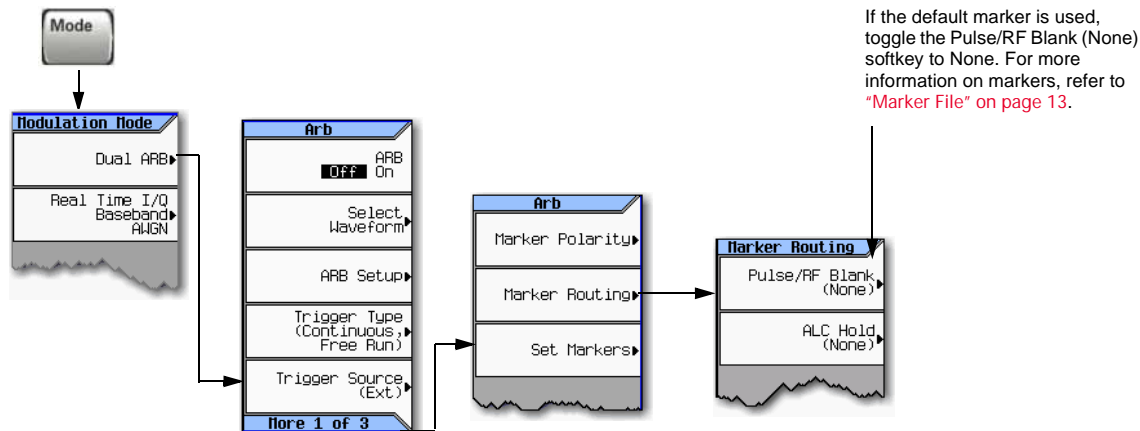
**Program Comments**

| 10: | Program file name. |
|---|---|
| 20 | Sets the number of points in the waveform. |
| 30: | Defines arrays for I and Q waveform points. Sets them to be integer arrays. |
| 40: | Sets HP BASIC to use degrees for cosine and sine functions. |
| 50: | Sets up loop to calculate waveform points. |
| 60: | Calculates I waveform points. |
| 70: | Calculates Q waveform points. |
| 80: | End of loop. |
| 160 and 210: | The I and Q waveform files have the same name |
| 90 to 300: | See the table on page 104 for program comments. |

### Creating and Downloading E443xB Waveform Data Using HP Basic for UNIX

On the documentation CD, this programming example's name is
*"e443xb_hpbasicUx2.txt."*

The following program shows you how to download waveforms using HP BASIC for UNIX. It is similar to the previous program example. The difference is the way the formatting for the most significant bit (MSB) on lines is handled.

First, the I waveform data is put into an array of integers called `Iwfm_data` and the Q waveform data is put into an array of integers called `Qwfm_data`. The variable `Nbytes` is set to equal the number of bytes in the I waveform data. This should be twice the number of integers in `Iwfm_data`, since an integer is represented 2 bytes. Input integers must be between 0 and 16383.

In the `Output` commands, `USING "#,K"` formats the data. The pound symbol (#) suppresses the automatic EOL (End of Line) output. This allows multiple output commands to be concatenated as if they were a single output. The "K" instructs HP BASIC to output the following numbers or strings in the default format.

```
10    ! RE-SAVE "ARB_IQ_file"

20    Num_points=200

30    ALLOCATE INTEGER
Iwfm_data(1:Num_points),Qwfm_data(1:Num_points)

40    DEG

50    FOR I=1 TO Num_points

60      Iwfm_data(I)=INT(8191*(SIN(I*360/Num_points))+8192)

70      Qwfm_data(I)=INT(8191*(COS(I*360/Num_points))+8192)

80    NEXT I

90    PRINT "Data Generated"

100   Nbytes=2*Num_points

110   ASSIGN @Esg TO 719;FORMAT ON

120   ASSIGN @Esgb TO 719;FORMAT OFF

130   Nbytes$=VAL$(Nbytes)

140   Ndigits=LEN(Nbytes$)

150   Ndigits$=VAL$(Ndigits)

160   OUTPUT @Esg USING "#,K";":MMEM:DATA ""ARBI:file_name_1""",#"

170   OUTPUT @Esg USING "#,K";Ndigits$

180   OUTPUT @Esg USING "#,K";Nbytes$

190   OUTPUT @Esgb;Iwfm_data(*)

200   OUTPUT @Esg;END

210   OUTPUT @Esg USING "#,K";":MMEM:DATA ""ARBQ:file_name_1""",#"

220   OUTPUT @Esg USING "#,K";Ndigits$

230   OUTPUT @Esg USING "#,K";Nbytes$

240   OUTPUT @Esgb;Qwfm_data(*)

250   OUTPUT @Esg;END

260   ASSIGN @Esg TO *

270   ASSIGN @Esgb TO *

280   PRINT

290   PRINT "*END*"
```

300     END

**Program Comments**

| 10: | Program file name. |
|---|---|
| 20 | Sets the number of points in the waveform. |
| 30: | Defines arrays for I and Q waveform points. Sets them to be integer arrays. |
| 40: | Sets HP BASIC to use degrees for cosine and sine functions. |
| 50: | Sets up loop to calculate waveform points. |
| 60: | Calculates I waveform points. |
| 70: | Calculates Q waveform points. |
| 80: | End of loop. |
| 160 and 210: | The I and Q waveform files have the same name |
| 90 to 300 | See the table on page 107 for program comments. |

# Troubleshooting Waveform Files

| Symptom | Possible Cause |
|---------|----------------|
| ERROR 224, Text file busy | Attempting to download a waveform that has the same name as the waveform currently being played by the signal generator.<br><br>To solve the problem, either change the name of the waveform being downloaded or turn off the ARB. |
| ERROR 628, DAC over range | The amplitude of the signal exceeds the DAC input range. The typical causes are unforeseen overshoot (DAC values within range) or the input values exceed the DAC range.<br><br>To solve the problem, scale or reduce the DAC input values. For more information, see "DAC Input Values" on page 8.<br><br>On the Keysight MXG, this error can occur if an encrypted file (`.SECUREWAVE`) is being downloaded to the signal generator from a PC or USB Media with a different suffix (i.e. **not** .SECUREWAVE).<br><br>To solve the problem, use the **Use as** or **Copy File to Instrument** softkey menus to download the encrypted file to the instrument. For more information, see "Encrypted I/Q Files and the Securewave Directory (Keysight MXG)" on page 27. |
| ERROR 629, File format invalid | The signal generator requires a minimum of 60 samples to build a waveform and the same number of I and Q data points. |
| ERROR −321, Out of memory | There is not enough space in the ARB memory for the waveform file being downloaded.<br><br>To solve the problem, either reduce the file size of the waveform file or delete unnecessary files from ARB memory. Refer to "Waveform Memory" on page 19. |
| No RF Output | The marker RF blanking function may be active. To check for and turn RF blanking off, refer to "Configuring the Pulse/RF Blank (Keysight MXG)" on page 113 and "Configuring the Pulse/RF Blank (ESG/PSG)" on page 113. This problem occurs when the file header contains unspecified settings and a previously played waveform used the marker RF blanking function.<br><br>For more information on the marker functions, see the *User's Guide*. |
| Undesired output signal | Check for the following:<br><br>— The data was downloaded in little endian order. See "Little Endian and Big Endian (Byte Order)" on page 6 for more information.<br><br>— The waveform contains an odd number of samples. An odd number of samples can cause waveform discontinuity. See "Waveform Phase Continuity" on page 16 for more information. |

## Configuring the Pulse/RF Blank (Keysight MXG)

If the default marker is used, toggle the Pulse/RF Blank (None) softkey to None. For more information on markers, refer to "Marker File" on page 13.

SCPI commands:

For details on each key, use the key help. Refer to *Programming Guide* and the *User's Guide*. For additional SCPI command information, refer to the *SCPI Command Reference.*

## Configuring the Pulse/RF Blank (ESG/PSG)

SCPI commands:

If the default marker is used, toggle the Pulse/RF Blank (None) softkey to None. For more information on markers, refer to "Marker File" on page 13.
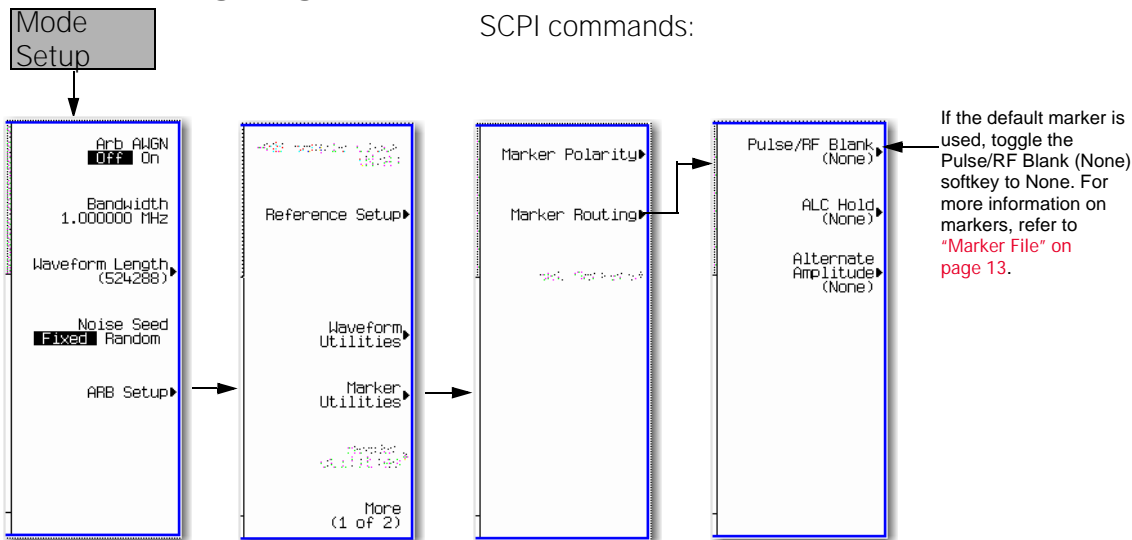
For details on each key, use the *Key and Data Field Reference.* For additional SCPI command information, refer to the *SCPI Command Reference.*

Index

**KEYSIGHT**
TECHNOLOGIES