CHAPTER FOUR: *Understand and Manage Waveforms*

In this chapter, see how

➢ *To learn how waveforms are structured*

➢ *To inspect waveform contents*

➢ *To transfer waveforms rapidly*

Know Your Waveform

A waveform can be said to have two main parts. The first is its basic data array: raw data values from the oscilloscope's ADCs (Analog-to-Digital Converters) obtained in the waveform's capture. The second is the description that accompanies this raw data: the vertical and horizontal scale or time of day, for example, necessary for a full understanding of the information contained in the waveform.

You can access this information by remote control using the `INSPECT?` query (see page 36), which interprets it in an easily understood ASCII text form. And you can rapidly transfer the information using the `WAVEFORM?` query (see page 37). Or write it back into the oscilloscope with the `WAVEFORM` command (page 42).

Your Waverunner oscilloscope contains a data structure, or template (see Appendix II), which provides a detailed description of how waveform information is organized. Although a sample template is provided with this manual, we suggest you use the `TEMPLATE?` query to access the Waverunner template in the oscilloscope itself (the template may change as your oscilloscope's firmware is enhanced).

You can also store waveforms in preformatted ASCII output, for popular spreadsheet and math processing packages, using the STORE and STORE_SETUP commands. Also see Chapter 12, "Use Waverunner with PC," of the *Operator's Manual*.

LOGICAL DATA BLOCKS

Each of your waveforms will normally contain at least a waveform descriptor and data array block. However, other blocks may also be present in more complex waveforms.

Waveform Descriptor block (WAVEDESC): This includes all the information necessary to reconstitute the display of the waveform from the data, including: hardware settings at the time of acquisition, the exact time of the event, kinds of processing performed, your oscilloscope name and serial number, the encoding format used for the data blocks, and miscellaneous constants.

Optional User-provided Text block (USERTEXT): Use the `WFTX` command to put a title or description of a waveform into this block, and the `WFTX?` query for an alternative way to read it. This text block can hold up to 160 characters. Display them as four lines of 40 characters by selecting "**Text & Times**" from the status menu, using Waverunner front panel controls (see the *Operator's Manual*).

Sequence Acquisition Times block (TRIGTIME): This is needed for sequence acquisitions to record the exact timing information for each segment. It contains the time of each trigger relative to the trigger of the first segment, as well as the time of the first data point of each segment relative to its trigger.

Random interleaved sampling times block (RISTIME): This is required for RIS acquisitions to record the exact timing information for each segment.

First data array block (SIMPLE or DATA_ARRAY_1): This is the basic integer data of the waveform. It can be raw or corrected ADC data or the integer result of waveform processing.

Second data array block (DATA_ARRAY_2): This is a second data array, needed to hold the results of processing functions such as Extrema or FFT math functions:

	EXTREMA	**FFT**
DATA_ARRAY_1	Roof trace	Real part
DATA_ARRAY_2	Floor trace	Imaginary part

NOTE: The Waverunner template also describes an array named DUAL. But this is simply a way to allow the `INSPECT?` command to examine the two data arrays together.

INSPECT WAVEFORM CONTENTS

Use the INSPECT? query to examine the contents of your waveform. You can use it on both of the main waveform parts. Its most basic form is: INSPECT? "name", the template giving you the name of a descriptor item or data block. The answer is returned as a single string, but may cover many lines. Some typical dialogue follows:

Question   C1:INSPECT? "VERTICAL_OFFSET"

Response   C1:INSP "VERTICAL_OFFSET: 1.5625e−03"

Question   C1:INSPECT? "TRIGGER_TIME"

Response   C1:INSP "TRIGGER_TIME: Date = FEB 17, 1994, Time = 4: 4:29.5580"

You can also use INSPECT? to provide a readable translation of the full waveform descriptor block using INSPECT? "WAVEDESC". Again, the template will give you the details for interpretation of each of the parameters. Use, too, INSPECT? "SIMPLE" to examine the measured data values of a waveform. For an acquisition with 52 points, for example:

```
INSPECT? "SIMPLE"
C1:INSP "
 0.0005225   0.0006475  −0.00029    −0.000915    2.25001E−0   0.000835
 0.0001475  −0.0013525  −0.00204    −4E−05       0.0011475    0.0011475
−0.000915   −0.00179    −0.0002275   0.0011475   0.001085    −0.00079
−0.00179    −0.0002275   0.00071      0.00096    −0.0003525  −0.00104
 0.0002725   0.0007725   0.00071     −0.0003525  −0.00129    −0.0002275
 0.0005225   0.00046    −0.00104     −0.00154     0.0005225   0.0012725
 0.001335   −0.0009775  −0.001915    −0.000165    0.0012725   0.00096
−0.000665   −0.001665   −0.0001025    0.0010225   0.00096    −0.0003525
−0.000915    8.50001E−0  0.000835     0.0005225
"
```

The numbers in the table above are the fully converted measurements in volts. When the data block contains thousands of items the string will contain a great many lines.

Depending on the application, you may prefer the data in its raw form, with either a BYTE (8 bits) or a WORD (16 bits) for each data value. In that case, use the relations INSPECT? "SIMPLE",BYTE with WAVEFORM?. The examination of data values for waveforms with two data arrays can be performed as follows:

INSPECT? "DUAL"   to get pairs of data values on a single line

INSPECT? "DATA_ARRAY_1"   to get the values of the first data array

INSPECT? "DATA_ARRAY_2"   to get the values of the second data array.

INSPECT? has its limitations; it is useful, but also wordy. As a query only, INSPECT? cannot be used to send a waveform back to the oscilloscope. If you want to do this and you want the information quickly, you should instead use WAVEFORM. With WAVEFORM_SETUP it is possible to examine just a part of the waveform or a sparsed form of it. See the following pages.

If you're a BASIC user you might also find it convenient to use INSPECT? and WAVEFORM? together to construct files containing a version of the waveform descriptor that both you and BASIC can read. Using a stored waveform, this can be done in a format suitable for retransfer to Waverunner with MC:INSPECT? "WAVEDESC";WAVEFORM?, and then placing the response directly into a disk file.

USE THE WAVEFORM QUERY

Use the WAVEFORM? query to transfer waveform data in block formats defined by the IEEE-488.2 standard. You can then download the response back to your Waverunner by using the WAVEFORM command. All your waveform's logical blocks can be read with the query C1:WAVEFORM? Completeness, as well as time and space are the advantages of this approach when you have to read many waveforms with the same acquisition conditions, or when you are interested only in large amounts of raw integer data. Moreover, you can choose any single block for reading with a query such as C1:WAVEFORM? DAT1. See Part Two for the various block names.

You can place the binary response to a query of the form C1:WAVEFORM? or C1:WAVEFORM? ALL in a disk file, then dump it using the GPIB bus. Do this with default settings to show the hexadecimal and ASCII form, as on the following page.

NOTE: A waveform query response can easily be a block containing over 16 million bytes if it is in binary format, and twice as much if the HEX option is used.

Byte Offset Number		Binary Contents in Hexadecimal	ASCII Translation (.... = uninteresting)
0		43 31 3A 57 46 20 41 4C 4C 2C 23 39 30 30 30 30	C1:WFALL,#90000
16		30 30 34 35 30	00450
	0	57 41 56 45 44 45 53 43 00 00 00	WAVEDESC...
32	11	00 00 00 00 00 4C 45 43 52 4F 59 5F 32 5F 32 00	.....LECROY_2_2.
48	27	00 00 00 00 00 00 01 00 00 00 00 01 5A 00 00 00	....
64	43	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
80	59	00 00 00 00 68 00 00 00 00 00 00 00 00 00 00 00	....
96	75	00 4C 45 43 52 4F 59 39 33 37 34 4C 00 00 00 00	.LECROYLT344....
112	91	00 37 84 09 40 00 00 00 00 00 00 00 00 00 00 00	
128	107	00 00 00 00 00 00 00 00 34 00 00 00 34 00 00 00	
144	123	32 00 00 00 00 00 00 33 00 00 00 00 00 00 00 00	
160	139	01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00	
176	155	00 34 83 12 6F 3A 0D 8E C9 46 FE 00 00 C7 00 00	
192	171	00 00 08 00 01 32 2B CC 77 BE 6B A4 BB 51 A0 69	
208	187	BB BE 6A D7 F2 A0 00 00 00 56 00 00 00 00 00 00	
224	203	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
240	219	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
256	235	00 00 00 00 00 00 00 00 00 00 53 00 00 00 00 00	
272	251	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
288	267	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
304	283	00 00 00 00 00 00 00 00 00 00 00 00 00 40 3B 00	
320	299	00 00 00 00 00 17 0A 05 02 07 C8 00 00 00 00 00	
336	315	00 00 00 00 00 00 00 00 01 00 0E 00 04 3F 80 00	
352	331	00 00 0A 00 00 3F 80 00 00 3A 0D 8E C9 00 00	
367	0	11	
368	1	00 13 00 04 00 FA 00 09 00 16 00 0B 00 F3 00 E8	
384	17	00 08 00 1B 00 1B 00 FA 00 EC 00 05 00 1B 00 1A	
400	33	00 FC 00 EC 00 05 00 14 00 18 00 03 00 F8 00 0D	
416	49	00 15 00 14 00 03 00 F4 00 05 00 11 00 10 00 F8	
432	65	00 F0 00 11 00 1D 00 1E 00 F9 00 EA 00 06 00 1D	
448	81	00 18 00 FE 00 EE 00 07 00 19 00 18 00 03 00 FA	
464	97	00 0A 00 16 00 11 00	
471	(Terminator)	0A	

Above: To illustrate the contents of the logical blocks, the relevant parts have been separated. To make counting easier, the corresponding Byte Offset numbering has been restarted each time a new block begins. The ASCII translation, only part of which is shown, has been similarly split and highlighted, showing how its parts correspond to the binary contents.

On the facing page... The first 10 bytes translate into ASCII and resemble the simple beginning of a query response. These are followed by the string #9000000450, the beginning of a binary block in which nine ASCII integers are used to give the length of the block (450 bytes). The waveform itself starts immediately after this, at Byte 21. The very first byte is 0, as it is for the first byte in each block.

The first object is a DESCRIPTOR_NAME, a string of 16 characters with the value WAVEDESC.

Then, 16 bytes after the beginning of the descriptor, at Byte 37, we find the beginning of the next string: the TEMPLATE_NAME with the value LECROY_2_2.

Several other parameters follow. The INSTRUMENT_NAME, LECROYLT344, 76 bytes from the descriptor start (Byte 97), is easily recognizable. On the preceding line, 38 bytes after the descriptor (Byte 59), a four-byte integer gives the length of the descriptor: WAVE_DESCRIPTOR = 00 00 01 5A (hex) = 346.

At 60 bytes from the descriptor start (Byte 81) we find another four-byte integer giving the length of the data array: WAVE_ARRAY_1 = 00 00 00 68 (hex) = 104.

And at 116 bytes after the descriptor (Byte 137), yet another four-byte integer gives the number of data points: WAVE_ARRAY_COUNT = 00 0000 34 (hex) = 52.

Now we know that the data will start at 346 bytes from the descriptor's beginning (Byte 367), and that each of the 52 data points will be represented by two bytes. The waveform has a total length of 346 + 104, which is the same as the ASCII string indicated at the beginning of the block. The final 0A at Byte 471 is the NL character associated with the GPIB message terminator <NL><EOI>.

As the example was taken using an oscilloscope with an eight-bit ADC, we see the eight bits followed by a 0 byte for each data point. However, for many other kinds of waveform this second byte will not be zero and will contain significant information. The data is coded in signed form (two's complement) with values ranging from -32768 = 8000 (hex) to 32767 = 7FFF (hex). If we had chosen to use the BYTE option for the data format, the values would have been signed integers in the range -128 = 80 (hex) to 127 = 7F (hex). The ADC values are mapped to the display grid in the following way:

➢ 0 is located on the grid's center axis

➢ 127 (BYTE format) or 32767 (WORD format) is located at the top of the grid

➢ -128 (BYTE format) or -32768 (WORD format) is located at the bottom of the grid.

INTERPRET VERTICAL DATA

Knowing now how to decipher the data, you may wish to convert it to the appropriate measured values. The vertical reading for each data point depends on the vertical gain and the vertical offset given in the descriptor. For acquisition waveforms, this corresponds to the volts/div and voltage offset selected after conversion for the data representation being used. The template tells us that the vertical gain and offset can be found at Bytes 156 and 160 and that they are stored as floating point numbers in the IEEE 32-bit format. An ASCII string giving the vertical unit is to be found in VERTUNIT, Byte 196. The vertical value is given by the relationship: value = VERTICAL_GAIN × data − VERTICAL_OFFSET, where:

VERTICAL_GAIN	2.44141e−07 from the floating point number 3483 126f at Byte 177
VERTICAL_OFFSET	0.00054 from the floating point number 3A0D 8EC9 at Byte 181
VERTICAL_UNIT	V = volts from the string 5600 ... at Byte 217

Therefore:

since data[4] = FA00 = 64000 from the hexadecimal word FA00 at byte 371. Overflows the maximum. 16 bit value of 32767, so must be a negative value. Using the two's complement conversion $64000 - 2^{16} = -1536$

value[4] = −0.000915 V as stated in the inspect command.

If the computer or the software available is not able to understand the IEEE floating point values, use the description in the template.

The data values in a waveform may not all correspond to measured points. FIRST_VALID_PNT and LAST_VALID_PNT give the necessary information. The descriptor also records the SPARSING_FACTOR, the FIRST_POINT, and the SEGMENT_INDEX to aid interpretation if the options of the WAVEFORM_SETUP command have been used.

For sequence acquisitions the data values for each segment are given in their normal order and the segments are read out one after the other. The important descriptor parameters are the WAVE_ARRAY_COUNT and the SUBARRAY_COUNT, giving the total number of points and the number of segments.

For waveforms such as the extrema and the complex FFT there will be two arrays — one after the other — for the two of the result.

CALCULATE A DATA POINT'S HORIZONTAL POSITION

Each vertical data value has a corresponding horizontal position, usually measured in time or frequency units. The calculation of this position depends on the type of waveform. Each data value has a position, i, in the original waveform, with i = 0 corresponding to the first data point acquired. The descriptor parameter HORUNIT gives a string with the name of the horizontal unit.

Single Sweep waveforms: x[i] = HORIZ_INTERVAL × i + HORIZ_OFFSET. For acquisition waveforms this time is from the trigger to the data point in question. It will be different from acquisition to acquisition since the HORIZ_OFFSET is measured for each trigger. In the case of the data shown above this means:

HORIZ_INTERVAL = 1e−08 from the floating point number 322b cc77 at Byte 194

HORIZ_OFFSET = −5.149e−08 from the double precision floating point number be6b a4bb 51a0 69bb at Byte 198

HORUNIT = S = seconds from the string 5300 ... at Byte 262

This gives: x[0] = −5.149e−08 S
x[1] = −4.149e−08 S.

40

Sequence waveforms: are really many independent acquisitions, so each segment will have its own horizontal offset. These can be found in the TRIGTIME array.

For the n'th segment:

$$x[i,n] = \text{HORIZ\_INTERVAL} \times i + \text{TRIGGER\_OFFSET}[n].$$

The TRIGTIME array can contain up to 200 segments of timing information with two eight-byte double precision floating point numbers for each segment.

RIS (Random Interleaved Sampling) waveforms: are composed of many acquisitions interleaved together. The descriptor parameter, RIS_SWEEPS gives the number of acquisitions. The i'th point will belong to the m'th segment where:

$$m = i \text{ modulo (RIS\_SWEEPS)} \text{ will have a value between 0 and RIS\_SWEEPS} -1.$$

Then with: $j = i - m$

$$x[i] = x[j,m] = \text{HORIZ\_INTERVAL} \times j + \text{RIS\_OFFSET}[m],$$

where the RIS_OFFSETs can be found in the RISTIME array. There can be up to 100 eight-byte double precision floating point numbers in this block. The instrument tries to get segments with times such that: $\text{RIS\_OFFSET}[i] \cong \text{PIXEL\_OFFSET} + (i - 0.5) \times \text{HORIZ\_INTERVAL}.$

Thus, taking as an example a RIS with RIS_SWEEPS = 10, HORIZ_INTERVAL = 1 ns, and PIXEL_OFFSET = 0.0, we might find for a particular event that:

RIS_OFFSET[0] = −0.5 ns	RIS_OFFSET[1] = 0.4 ns
RIS_OFFSET[2] = 1.6 ns	RIS_OFFSET[3] = 2.6 ns
RIS_OFFSET[4] = 3.4 ns	RIS_OFFSET[5] = 4.5 ns
RIS_OFFSET[6] = 5.6 ns	RIS_OFFSET[7] = 6.4 ns
RIS_OFFSET[8] = 7.6 ns	RIS_OFFSET[9] = 8.5 ns

and therefore:

x[0] =	RIS_OFFSET[0] = −0.5 ns
x[1] =	RIS_OFFSET[1] = 0.4 ns
...	
x[9] =	RIS_OFFSET[9] = 8.5 ns
x[10] =	1 ns × 10 + (−0.5) = 9.5 ns
x[11] =	1 ns × 10 + 0.4 = 10.4 ns
...	
x[19] =	1 ns ´ 10 + 8.5 = 18.5 ns
x[20] =	1 ns ´ 20 + (−0.5) = 19.5 ns.
...	

USE THE WAVEFORM COMMAND

Waveforms you read with the WAVEFORM? query (page 37) can be sent back into your Waverunner oscilloscope using WAVEFORM and related commands. Since the descriptor contains all of the necessary information, you need not be concerned with any of the communication format parameters. The oscilloscope will learn all it needs to know from the waveform.

> **TIP: Because waveforms can only be sent back to Waverunner memory traces (M1, M2, M3, M4), consider removing or changing the prefix (C1 or CHANNEL_1) in the response to the WF? query. See Part Two for examples.**

To ensure that the descriptor is coherent, however, when you synthesize waveforms for display or comparison read out a waveform of the appropriate size and then replace the data with the desired values.

Here are among the many ways to use WAVEFORM and its related commands to simplify or speed up work:

Partial Waveform Readout: Use WAVEFORM_SETUP to specify a short part of a waveform for readout, as well as to select a sparsing factor for reading every n'th data point only.

Byte Swapping: The COMM_ORDER command allows you to swap two bytes of data presented in 16-bit word format, in the descriptor or in the the data/time arrays, when sending the data via GPIB or RS-232-C ports. Depending on the computer system used, this will allow easier data interpretation. For Intel-based computers, you should send the data with the LSB first; the command should be CORD LO. For Motorola-based computers, send the data with the MSB first (CORD HI) — the default at power-up.

> **NOTE: Data written to Waverunner hard disk or floppy drive, or to the optional PC memory card drive, will always remain in LSB first, the default DOS format. Thus you cannot use the CORD command in these cases, as it is only for data sent via the GPIB and RS-232-C ports.**

Data Length, Block Format, and Encoding: COMM_FORMAT gives you control over these parameters. If you do not need the extra precision of the lower order byte of the standard data value, the BYTE option will enable you to save by a factor of two the amount of data transferred or stored. If the computer you are using cannot read binary data, the HEX option allows a response form in which the value of each byte is given by a pair of hexadecimal digits.

Data-Only Transfers: COMM_HEADER OFF enables a response to WF? DAT1 with data only (the C1:WF DAT1 will disappear). If you have also specified COMM_FORMAT OFF,BYTE,BIN, the response will be data bytes only (the #90000nnnnn will disappear — see page 38).

Formatting for RS-232-C Users: The COMM_RS232 command can assist by splitting the very long WF? response into individual lines.

Transfer Waveforms at High Speed

You must take several important factors into account if you wish to achieve maximum, continuous data transfer rates from your Waverunner oscilloscope to the external controller. The single most important of these is to limit the amount of work done in the computer. This means that you should avoid writing data to disk wherever possible, minimize operations such as per-data-point computations, and reduce the number of calls to the I/O system. To do this, you can try the following:

➤ **Reduce the number of points to be transferred and the number of data bytes per point.** The pulse parameter capability and the processing functions can save a great deal of computing and a lot of data transfer time if employed creatively.

➤ **Attempt to overlap waveform acquisition with waveform transfer.** The oscilloscope is capable of transferring an already acquired or processed waveform after a new acquisition has been started. The total time that Waverunner takes to acquire events will be considerably increased if it is obliged to wait for triggers (livetime).

➤ **Minimize the number of waveform transfers by using Sequence mode** to accumulate many triggers for each transfer. This is preferable to using `WAVEFORM_SETUP` to reduce the number of data points for transfer. It also significantly reduces oscilloscope transfer overhead. For example, you could use `ARM; WAIT;C1:WF?` (wait for the event, transfer the data, and then start a new acquisition). You could also "loop" this line in the program as soon as it has finished reading the waveform.