Keysight Signal Extractor Tool

Online Help

KEYSIGHT TECHNOLOGIES

Notices

Trademarks

Microsoft®, MS-DOS®, Windows®, Windows 2000®, and Windows XP® are U.S. registered trademarks of Microsoft Corporation.

Revision History

August 2014

Available in electronic format only

Keysight Technologies
1900 Gardens of the Gods Road
Colorado Springs, CO 80907 USA

Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Keysight Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

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.

Using the Signal Extractor Tool

The Signal Extractor is a simple tool that lets you extract signals or data from one input bus/signal and output the extracted data onto new buses/signals. In particular, it is suited to:

- Extract I and Q data (see page 9) from simple serial protocols.
- Remultiplex high-speed digital data (see page 13) that has been demultiplexed onto additional logic analyzer channels (because of the high state speeds).
- Convert timing analysis captures of serial buses into samples that can be decoded and displayed using the Packet Decoder and Packet Viewer. See **"Using the FindPulseWidth Command"** on page 31 and **Chapter 1**, “What's New in the Signal Extractor Tool,” starting on page 7.

Once you add a Signal Extractor tool, its properties dialog opens to let you:

- Select the input bus/signal.
- Load an XML format extractor algorithm file.

The extractor algorithm file defines output buses/signals and specifies how data should be extracted by identifying patterns and listing the commands to execute when those patterns are found.

- Signal Extractor Capabilities (see page 15)
- How the Signal Extractor Tool Works (see page 17)
- Adding a Signal Extractor Tool (see page 19)
- Creating Extractor Algorithms (see page 21)
- Signal Extractor Tool Properties Dialog (see **page 59**)
- Signal Extractor Tool Control, COM Automation (see **page 61**)
- Signal Extractor Tool Setup, XML Format (see **page 63**)

Contents

1 What's New in the Signal Extractor Tool

Version 03.80 of the *Keysight Logic Analyzer* application provides these additions to the Signal Extractor tool:

New Signal Extractor Commands

Command	Description
FindPulseWidth (see page 52)	Calculates the smallest pulse width found on a specified input channel. See "Using the FindPulseWidth Command" on page 31.
JumpCase1BitReg (see page 53)	Equivalent to the `JumpCase1Bit` command except that the bit is loaded indirectly through a register.
JumpCase2BitReg (see page 53)	Equivalent to the `JumpCase2Bit` command except that the bits are loaded indirectly through registers.
JumpCase3BitReg (see page 53)	Equivalent to the `JumpCase3Bit` command except that the bits are loaded indirectly through registers.
JumpCase4BitReg (see page 54)	Equivalent to the `JumpCase4Bit` command except that the bits are loaded indirectly through registers.
JumpTimeGreaterEqualRegs (see page 54)	Equivalent to the `JumpTimeGreaterEqual` command except that the bits whose timestamps are used are specified by register values.
LoadBitReg (see page 55)	Equivalent to the `Load` command except that the bit to load is specified by the value in a register.
MovReg (see page 56)	Loads a register with an unsigned 32-bit integer value. This replaces the `LoadReg` command.
Mov2Regs (see page 56)	Loads one register with a value from another register.
WriteLabelTimeDeltaRegs (see page 57)	Equivalent to the `WriteLabelTimeDelta` command except that the bits whose timestamps are used are specified by register values.
WriteLabelTimeReg (see page 57)	Equivalent to the `WriteLabelTime` command except that the bit whose timestamp is used is specified by a register value.

Other Changes

The number of registers has been changed from 4 to 16. Register 0 is still the 128-bit write register. Registers 1-15 are available as 32-bit registers.

2 Example: Extracting I and Q Signals

Here we are extracting the I and Q signals from a simple serial stream, serial protocol, or from a parallel bus.

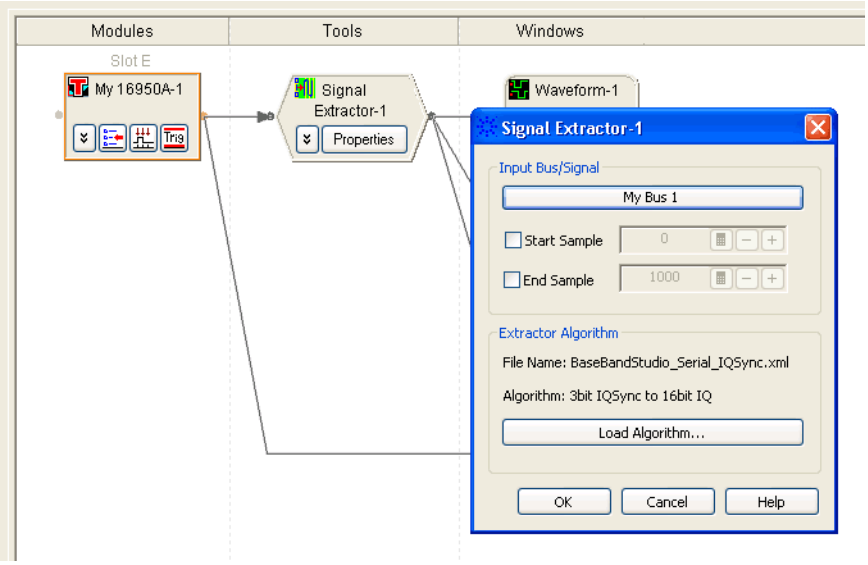
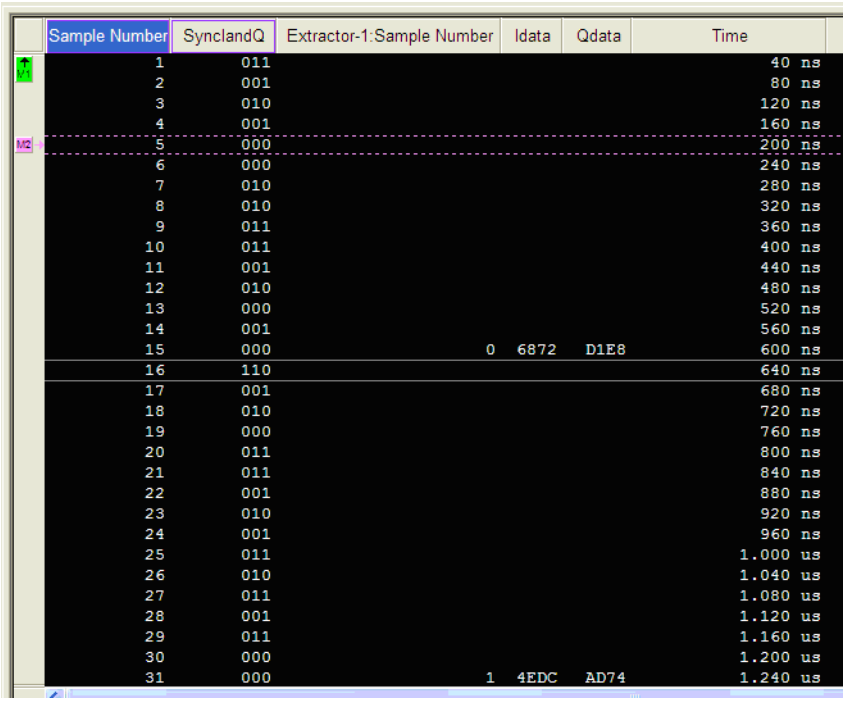


Figure 1 Signal Extractor Properties

The I and Q signals (in this example 16-bit wide buses) can then be viewed with other logic analyzer tools.



Sample Number	SynclandQ	Extractor-1:Sample Number	Idata	Qdata	Time
1	011				40 ns
2	001				80 ns
3	010				120 ns
4	001				160 ns
5	000				200 ns
6	000				240 ns
7	010				280 ns
8	010				320 ns
9	011				360 ns
10	011				400 ns
11	001				440 ns
12	010				480 ns
13	000				520 ns
14	001				560 ns
15	000		0	6872 D1E8	600 ns
16	110				640 ns
17	001				680 ns
18	010				720 ns
19	000				760 ns
20	011				800 ns
21	011				840 ns
22	001				880 ns
23	010				920 ns
24	001				960 ns
25	011				1.000 us
26	010				1.040 us
27	011				1.080 us
28	001				1.120 us
29	011				1.160 us
30	000				1.200 us
31	000		1	4EDC AD74	1.240 us

Figure 2 Signal extraction of 16-bit serial I and Q

For example, you can view the bus data as a chart in the Waveform display window.

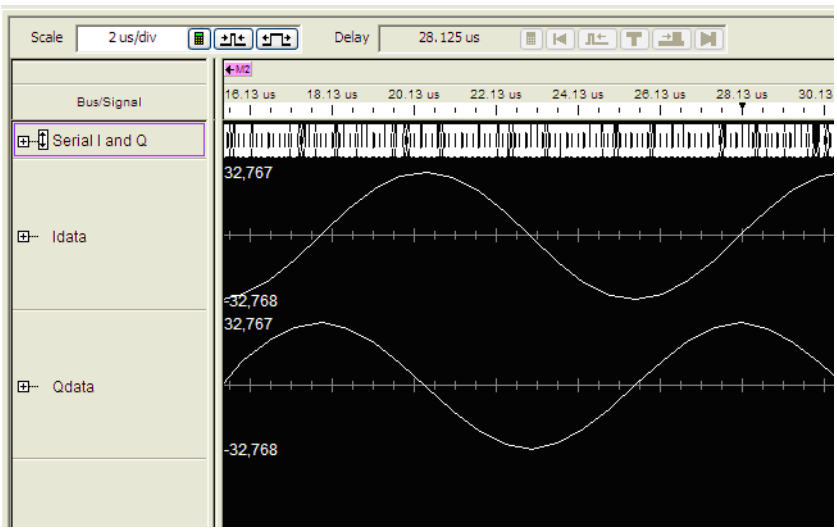


Figure 3 Second view of the extracted I and Q signals using chart

The I and Q samples can then be further processed to determine error vector magnitude (EVM) and a variety of other communications type measurements.

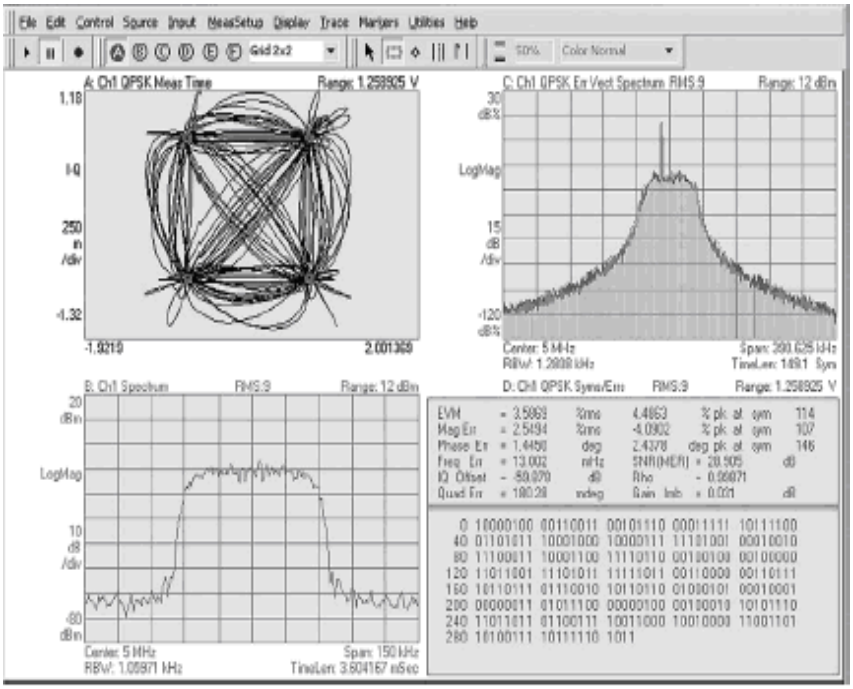


Figure 4 Keysight 89601A vector signal analysis software

3 Example: Remultiplex Data

Here we are remultiplexing high-speed digital data that has been demultiplexed onto additional logic analyzer channels (because of the high state speeds).

-1:Sample Number	Muxed Input	Time	-1:Sample Number	A-to-D
-9	42CA 42CA 42CA 42CA	-22.5 ns		
-8	42CB 42CB 42CB 42CB	-20.0 ns		
-7	42CC 42CC 42CC 42CC	-17.5 ns		
-6	42CD 42CD 42CD 42CD	-15.0 ns		
M1 -5	42CE 42CE 42CE 42CE	-12.5 ns		
-4	42CF 42CF 42CF 42CF	-10.0 ns		
-3	42D0 42D0 42D0 42D0	-7.5 ns		
-2	42D1 42D1 42D1 42D1	-5.0 ns		
-1	42D2 42D2 42D2 42D2	-2.5 ns		
T 0	42D3 42D3 42D3 42D3	0 s	0	42D3
		600 ps	1	42D3
		1.3 ns	2	42D3
		1.9 ns	3	42D3
1	42D4 42D4 42D4 42D4	2.5 ns	4	42D4
		3.1 ns	5	42D4
		3.8 ns	6	42D4
		4.4 ns	7	42D4
2	42D5 42D5 42D5 42D5	5.0 ns	8	42D5
		5.6 ns	9	42D5
		6.3 ns	10	42D5
		6.9 ns	11	42D5
3	42D6 42D6 42D6 42D6	7.5 ns	12	42D6
		8.1 ns	13	42D6
		8.8 ns	14	42D6
		9.4 ns	15	42D6
4	42D7 42D7 42D7 42D7	10.0 ns	16	42D7
		10.6 ns	17	42D7
		11.3 ns	18	42D7
		11.9 ns	19	42D7
M2 5	42D8 42D8 42D8 42D8	12.5 ns	20	42D8
		13.1 ns	21	42D8
		13.8 ns	22	42D8
		14.4 ns	23	42D8

Figure 5 Signal Extractor doing remultiplex starting at sample 0

4 Signal Extractor Capabilities

To understand the capabilities of the Signal Extractor consider these scenarios:

- You have a 4-bit wide serial bus with the following signals: 1-bit I value, 1-bit Q value, 1-bit sync signal, and 1-bit clock signal. The I and Q signals are each 16-bits long, in a serial manner, with the most significant bit of each I and Q 16-bit value coming on the occurrence of a sync signal.

  The Signal Extractor can look for the occurrence of the sync signal and then extract the serial I and serial Q values and write them as a pair of standard 16-bit wide I and Q values.

- You have a 2-bit wide serial bus with the following signals: 1-bit wide protocol encoded I and Q values, and 1-bit clock. Assume the packet header for I and Q values is a constant pattern followed by 4 pairs of 12-bit long I and Q values.

  The Signal Extractor can extract the 4 I and Q values and assign them to new buses. Again, the serial version of the I and Q values will be translated to pair of standard 12-bit wide I and Q values.

- You have a 48-bit wide bus with the following signals: 4x12 bits of A-to-D conversion. Each of the 12-bits is a high-speed sample of a 3.2 GHz A-to-D converter. The A-to-D converter is running at 3.2 MHz or 312 ps per sample. The signals are multiplexed out in groups of 4x12-bits and latched so that the logic analyzer can capture the A-to-D output at a rate of only 800 MHz or 1.25 ns per four samples.

  The Signal Extractor can remultiplex the A-to-D signal back to the original 12-bit wide signal.

5 How the Signal Extractor Tool Works

The Signal Extractor tool operates on one input bus/signal (which can be 1 to 128 bits wide). It operates on all data captured on that bus/signal unless starting and ending samples are specified.

The Signal Extractor tool extracts data from the input bus/signal using an algorithm that defines output buses/signals and specifies how data should be extracted. The extraction algorithm is based upon a series of ASCII, XML-format commands defined in a single algorithm file. You can create your own extraction algorithms or customize existing extraction algorithms. The extraction algorithm is based on a "find pattern then execute commands" model.

The Signal Extractor runs by looking for matching patterns from 1 to 128 bits in sequence. When a pattern is found, Signal Extractor commands are executed to extract out the specified signals. When the end of the extractor commands is reached, the Signal Extractor repeats the cycle by looking for matching patterns again from the beginning of the list.

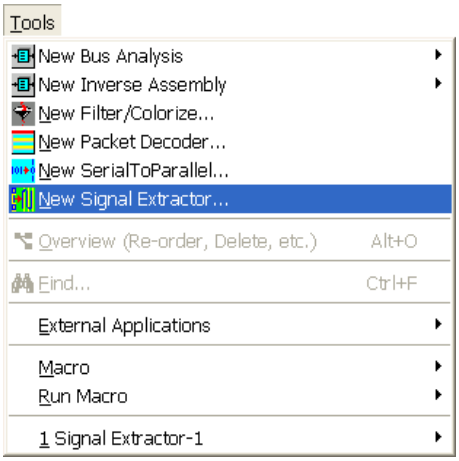
The Signal Extractor tool runs every time new data is received, and it runs completely through the entire data input and outputs a complete set of extracted data.

Up to a total of 8 buses/signals (or columns of data) can be extracted from a single bus/signal. The eight output buses/signals are actually split into 2 groups of four buses/signals each. These sets of four buses/signals each have their own time base and sample count. For example, you can extract I and Q data as a pair of buses/signals with one time base and sample count. Frame start markers and perhaps control commands can be extracted into the second set of buses/signals with a second time base and second sample count.

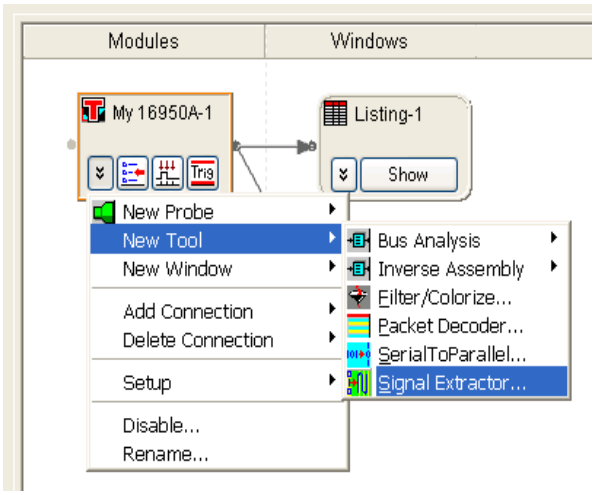
6 Adding a Signal Extractor Tool

This procedure assumes that a bus or signal that has data to be extracted has already been defined in the logic analyzer module (see "Defining Buses and Signals" (in the online help)).

1 From the main menu, choose Tools>New Signal Extractor....



Or, in the Overview window, from a logic analyzer module's drop-down menu, choose New Tool>Signal Extractor....



2 In the Signal Extractor dialog (see page 59), set up the tool properties:

a In the Input Bus/Signal box, click the bus button and select the name of the bus/signal that you want to extract data from.

You can extract signals from one input bus or signal (1 to 128 bits wide).

If your logic analyzer has a large memory depth and you want to reduce the number of samples that are processed (and speed up the processing of data), specify the Start Sample and End Sample of the data you want to process.

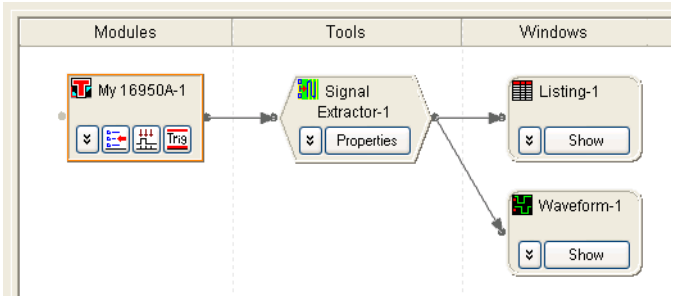
b In the Extractor Algorithm box, click Load Algorithm... and select the extractor algorithm file (see Creating Extractor Algorithms (see page 21)); then, click Open.

Example extractor algorithms have been included. You can find them under "Documents and Settings" for "All Users". For example, the default location is:

C:\Documents and Settings\All Users\Shared Documents\Keysight Technologies\Logic Analyzer\Extractor Algorithms

3 Click OK to close the Signal Extractor dialog and perform the extraction.

In the Overview window, the tool connection will look something like:



See Also · Chapter 7, “Creating Extractor Algorithms,” starting on page 21

7 Creating Extractor Algorithms

A typical extraction algorithm file has this format:

- Algorithm name.
- Output bus/signal definitions.
- Signal Extractor patterns.
- Signal Extractor commands.

Extractor algorithm files are ASCII XML format files which you can edit with any text editor.

XML format files and strings have markup like `<Tag>data</Tag>` where `Tag` is an element name, `<Tag>` is a start tag, and `</Tag>` is an end tag.

XML elements can have data or child elements; child elements can have data or children, and so on.

XML elements can have attributes, which are *name='value'* pairs within the element's start tag. Values must be contained in quotes.

Empty tags are used for elements that don't have any data. Empty tags are distinguished from start tags by a closing `/>` instead of a closing `>`. Empty tags can have attributes just like ordinary start tags.

Tag and attribute names are case sensitive (for example, "width" is not same as "Width").

To create an extractor algorithm file:

1 Start with the `<ExtractorGrammar>` (see page 46) start and end tags:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
</ExtractorGrammar>
```

The `AlgorithmDescription` attribute is a text string that gives the algorithm a name; this string also appears in the Signal Extractor tool's properties dialog when the file is loaded.

2 Next, define the output buses/signals using `<ExtractorLabel>` (see page 47) elements within `<ExtractorLabels>` start and end tags:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
   </ExtractorLabels>
</ExtractorGrammar>
```

You can define up to four output buses/signals on one timebase. If you want to define output signals on a different timebase, use the `<ExtractorFolder>` (see page 46) start and end tags which can contain up to four additional `<ExtractorLabel>` elements:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
```

```
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
</ExtractorGrammar>
```

Some rules for <ExtractorLabel> and <ExtractorFolder> elements:

- Only one <ExtractorFolder> is allowed. You can generate up to 4 buses/signals outside of a folder, and up to 4 buses/signals inside a folder. You can only generate two time bases.
- Bus/signal names must be unique inside or outside the folder (that is, you cannot have a signal named "Valid" outside the folder and another one inside the folder).
- The first bus/signal named in <ExtractorLabels> must be the first one written to, and it must be written to with the WriteLabelTime (see page 57) command. This is also true for the first bus/signal under <ExtractorFolder>.
- At least one <ExtractorLabel> element must be outside of the <ExtractorFolder> element; otherwise, there will be an error when loading the extractor algorithm.

3 Next, define patterns to look for and commands to execute for those patterns within <ExtractorSequences> and <ExtractorSequence> start and end tags:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
      </ExtractorSequence>
   </ExtractorSequences>
</ExtractorGrammar>
```

Note that the <ExtractorSequences> element can contain multiple <ExtractorSequence> elements. This means you can look for multiple patterns and have a unique command sequence for each pattern. However, only one command sequence can be active at a time. The first pattern that matches will cause the Signal Extractor to start executing the associated command sequence.

a To define patterns to look for, use <ExtractorPattern> (see page 48) elements within <ExtractorPatterns> start and end tags:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial'
 >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
```

```
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='b1XX' Width='3' Enabled='T' /
>
         </ExtractorPatterns>
      </ExtractorSequence>
   </ExtractorSequences>
</ExtractorGrammar>
```

Pattern value masks can be 1 to 128 bits wide and do not have to be equal to the size of the input bus/signal. This lets you find long (up to 128-bit) patterns on a single bit wide input signal. Pattern value masks can accept 1's, 0's, and don't cares.

b To define commands to execute when a pattern is found, use `<ExtractorCmd>` (see page 43) elements within `<ExtractorCmds>` start and end tags:

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial'
 >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='b1XX' Width='3' Enabled='T' /
>
         </ExtractorPatterns>
         <ExtractorCmds>
            <ExtractorCmd Cmd='Load' Bit='2' />
            <ExtractorCmd Cmd='Load' Bit='5' />
            <ExtractorCmd Cmd='Load' Bit='8' />
            <ExtractorCmd Cmd='Load' Bit='11' />
            <ExtractorCmd Cmd='Load' Bit='14' />
            <ExtractorCmd Cmd='Load' Bit='17' />
            <ExtractorCmd Cmd='Load' Bit='20' />
            <ExtractorCmd Cmd='Load' Bit='23' />
            <ExtractorCmd Cmd='Load' Bit='26' />
            <ExtractorCmd Cmd='Load' Bit='29' />
            <ExtractorCmd Cmd='Load' Bit='32' />
            <ExtractorCmd Cmd='Load' Bit='35' />
            <ExtractorCmd Cmd='Load' Bit='38' />
            <ExtractorCmd Cmd='Load' Bit='41' />
            <ExtractorCmd Cmd='Load' Bit='44' />
            <ExtractorCmd Cmd='Load' Bit='47' />
            <ExtractorCmd Cmd='WriteLabelTime' Name='Idata'
                  BitTime='0' />
            <ExtractorCmd Cmd='Load' Bit='1' />
            <ExtractorCmd Cmd='Load' Bit='4' />
            <ExtractorCmd Cmd='Load' Bit='7' />
            <ExtractorCmd Cmd='Load' Bit='10' />
```

```
            <ExtractorCmd Cmd='Load' Bit='13' />
            <ExtractorCmd Cmd='Load' Bit='16' />
            <ExtractorCmd Cmd='Load' Bit='19' />
            <ExtractorCmd Cmd='Load' Bit='22' />
            <ExtractorCmd Cmd='Load' Bit='25' />
            <ExtractorCmd Cmd='Load' Bit='28' />
            <ExtractorCmd Cmd='Load' Bit='31' />
            <ExtractorCmd Cmd='Load' Bit='34' />
            <ExtractorCmd Cmd='Load' Bit='37' />
            <ExtractorCmd Cmd='Load' Bit='40' />
            <ExtractorCmd Cmd='Load' Bit='43' />
            <ExtractorCmd Cmd='Load' Bit='46' />
            <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
            <ExtractorCmd Cmd='JumpDone' />
        </ExtractorCmds>
    </ExtractorSequence>
</ExtractorSequences>
</ExtractorGrammar>
```

When the pattern matches, the extractor command sequence phase starts. The command sequence phase can:

- Extract-bits into a buffer using these commands:
  - Load (see page 54)
  - LoadRange (see page 55)
- Insert 0's or 1's into a buffer using these commands:
  - LoadZero (see page 56)
  - LoadOne (see page 55)
- Use registers for counters and other math operations with these commands:
  - LoadInit (see page 55) to clear register 0
  - For other register commands, see "Using Register Commands" on page 27
- Compare bits or time between bits and jump forward or backward in the command sequence using these commands:
  - JumpCase1Bit (see page 53)
  - JumpCase1BitReg (see page 53)
  - JumpCase2Bit (see page 53)
  - JumpCase2BitReg (see page 53)
  - JumpCase3Bit (see page 53)
  - JumpCase3BitReg (see page 53)
  - JumpCase4Bit (see page 53)
  - JumpCase4BitReg (see page 54)
  - JumpForward (see page 54)
  - JumpBackward (see page 53)
  - JumpCmpReg (see page 54)
  - JumpCmp2Regs (see page 54)
  - JumpTimeGreaterEqual (see page 54)
  - JumpTimeGreaterEqualRegs (see page 54)

  Be careful not to create loops that never exit.

- Write the buffer to 1 of 8 buses/signals with a time tag or with a generated time tag, like: TimeTag1 + 1/4 (TimeTag2 - TimeTag1) using these commands:
  - WriteLabelTime (see page 57)
  - WriteLabelTimeDelta (see page 57)
  - WriteLabel (see page 57)
- Skip ahead X-bits of data using these commands:
  - GoTo (see page 53)
  - GoToReg (see page 53)
- Reset the zero location using this command:
  - ResetBitZero (see page 56)
- End the command sequence phase and return to looking for patterns using this command:
  - JumpDone (see page 54)
- Enable or disable extractor pattern masks using these commands:
  - EnablePattern (see page 52)
  - DisablePattern (see page 52)
- Split an input pattern into 2, 4, or 8 output patterns using this command:
  - Split (see page 56)

When the command sequence phase ends, the Signal Extractor returns to looking for pattern masks, starting again with the first pattern mask.

4 Note that you can include comments in the extractor algorithm file beginning with `<!--` and ending with `-->`. Comments can contain any text except double hyphen strings (which identify the beginning and the end of the comment). Comments can appear anywhere except inside tags. Comments can be used to hide XML elements. Beware of comments that hide required start or end tags.

```
<ExtractorGrammar AlgorithmDescription='Remultiplex Data Example' >
   <ExtractorLabels>
      <!-- Comment to describe output bus/signal names. -->
      <ExtractorLabel Name='A-to-D' Width='16' DefaultBase='Hex'
            VSAOutput='F'/>
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='hXXXXXXXXXXXXXXXX' Width='64'
                  Enabled='T' />
         </ExtractorPatterns>
         <ExtractorCmds>
            <!-- Comment to hide an element.
            <ExtractorCmd Cmd='Split' Amount='8' Size='8'
                  Name='A-to-D' />
            -->
            <ExtractorCmd Cmd='Split' Amount='4' Size='16'
                  Name='A-to-D' />
         </ExtractorCmds>
      </ExtractorSequence>
   </ExtractorSequences>
</ExtractorGrammar>
```

See Also · "Using Register Commands" on page 27

- “Using the FindPulseWidth Command" on page 31
- “Using Debug Mode" on page 41
- “Extractor Algorithm XML Format" on page 42
- “Signal Extractor Commands" on page 52
- Chapter 6, “Adding a Signal Extractor Tool,” starting on page 19
- The included example extractor algorithm files. You can find them under "Documents and Settings" for "All Users". For example, the default location is:

  C:\Documents and Settings\All Users\Shared Documents\Keysight Technologies\Logic Analyzer\Extractor Algorithms

Using Register Commands

There are various types of register commands:

- Register immediate operations:
  - MovReg (see page 56)
  - LoadReg (see page 55) (obsolete, replaced by MovReg)
  - AddReg (see page 52)
  - SubReg (see page 56)
  - MultReg (see page 56)
  - DivReg (see page 52)
  - AndReg (see page 52)
  - OrReg (see page 56)
  - AddRegSignedLimit (see page 52)
- Register immediate jump:
  - JumpCmpReg (see page 54)
- Two-register operations:
  - Mov2Regs (see page 56)
  - Add2Regs (see page 52)
  - Sub2Regs (see page 56)
  - Mult2Regs (see page 56)
  - Div2Regs (see page 52)
  - And2Regs (see page 52)
  - Or2Regs (see page 56)
  - Add2RegsSignedLimit (see page 52)
- Two-register jump:
  - JumpCmp2Regs (see page 54)
- Special register commands:
  - GoToReg (see page 53)
  - JumpCase1BitReg (see page 53)
  - JumpCase2BitReg (see page 53)
  - JumpCase3BitReg (see page 53)
  - JumpCase4BitReg (see page 54)
  - JumpTimeGreaterEqualRegs (see page 54)
  - LoadBitReg (see page 55)
  - LoadRangeRegs (see page 55)
  - WriteLabelTimeReg (see page 57)
  - WriteLabelTimeDeltaRegs (see page 57)

Register comments:

- Register 0 is an unsigned 128-bit integer and is the only register that can write to labels.
- Register 0 is automatically cleared after each write to a label.
- Register 0 should not be used to store values in.
- All math with register 0 converts results to unsigned 32-bit integers.
- Registers 1 through 15 are unsigned 32-bit integers and do unsigned 32-bit integer math.
- The JumpCmp commands use a 32-bit integer compare and jump to one of three locations.

- Divide will check for divide by 0 and make it divide by one so no exception will occur.
- The GoToReg is the same as GoTo Bit='x' except that we GoTo the bit position specified by the register. Only positive numbers. We cannot go to a negative number bit position.

To move the contents of register 0

You can move the contents of register 0 to a different register by using the Add2Regs (see page 52) command. For example:

```
<!-- Initialize register 1 to zero. -->
<ExtractorCmd Cmd='LoadReg' Number='1' Value='0' />

<!-- Load into register 0 bits zero through five. -->
<ExtractorCmd Cmd='LoadRange' BitStart='0' BitEnd='5' />

<!-- Move register zero to register one. -->
<ExtractorCmd Cmd='Add2Regs' Number='1' Second='0' />
```

To use registers 1-15 as counters

You can use the additional registers as counters. For example:

```
<!-- Initialize register 1 to five. -->
<ExtractorCmd Cmd='LoadReg' Number='1' Value='5' />

<!-- Subtract a value of 1 from register 1. -->
<ExtractorCmd Cmd='SubReg' Number='1' Value='1' />
<ExtractorCmd Cmd='JumpCmpReg' Number='1' Value='0' />

<!-- If less than zero. -->
<ExtractorCmd Cmd='JumpForward' Amount='10' />

<!-- If equal zero. -->
<ExtractorCmd Cmd='JumpForward' Amount='10' />

<!-- If greater than zero. -->
<ExtractorCmd Cmd='WriteLabelTime' Name='Data' BitTime='0' />
```

Example: using register commands for a long packet

```
<!-- Pick up the length. -->
<ExtractorCmd Cmd='LoadRange' ... />

<!-- Register 1 has the length. -->
<ExtractorCmd Cmd='AddReg' Number='1' Second='0' />

<!-- Make sub packets of 64-bytes. -->
<ExtractorCmd Cmd='SubReg' Number='1' Value='64' />
<ExtractorCmd Cmd='JumpCmpReg' Number='1' Value='0' />
<ExtractorCmd Cmd='JumpForward' Amount='20' /> <!-- to Less_End -->
<ExtractorCmd Cmd='JumpForward' Amount='10' /> <!-- to Equal_End -->
<ExtractorCmd Cmd='ResetBitZero' />

<!-- Do normal commands to output a sub packet of 64-bytes. -->
<ExtractorCmd Cmd='LoadReg' Number='0' Value='0' />
<ExtractorCmd Cmd='LoadRange' ... />
<ExtractorCmd Cmd='WriteLabel' ... />

<!-- Accumulate checksum. -->
<ExtractorCmd Cmd='Add2Regs' Number='2' Second='0' />
<ExtractorCmd Cmd='JumpBackward' ... /> <!-- to SubReg, repeat loop -->

<!-- Equal_End -->
```

Example: algorithm to demonstrate register usage

```
<!-- Less_End -->

<ExtractorGrammar AlgorithmDescription="Show off the register cmds" >
   <Comment Value='

      This simple demo shows how we can use registers.
      Run the analyzer off line with dummy data using an eight
      channel label.

   '/>
   <ExtractorLabels>
      <ExtractorLabel Name='Data' Width='8' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-2'/>
      <ExtractorLabel Name='Frame' Width='1' DefaultBase='Hex' />
      <ExtractorLabel Name='BigFrame' Width='1' DefaultBase='Hex' />
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='b1' Width='1' Enabled='T' />
         </ExtractorPatterns>
         <ExtractorCmds>
            <ExtractorCmd Cmd='LoadRange' BitStart='0' BitEnd='7'/>
            <ExtractorCmd Cmd='JumpCmpReg' Number='0' Value='h8' />
            <ExtractorCmd Cmd='JumpForward' Amount='35' />
            <ExtractorCmd Cmd='JumpForward' Amount='1' />
            <ExtractorCmd Cmd='JumpCmpReg' Number='0' Value='h22' />
            <ExtractorCmd Cmd='JumpForward' Amount='3' />
            <ExtractorCmd Cmd='JumpForward' Amount='2' />
            <ExtractorCmd Cmd='LoadReg' Number='0' Value='h23' />
            <ExtractorCmd Cmd='LoadReg' Number='1' Value='0' />
            <ExtractorCmd Cmd='Add2Regs' Number='1' Second='0' />
            <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                  BitTime='0' />
            <ExtractorCmd Cmd='LoadOne' />
            <ExtractorCmd Cmd='WriteLabel' Name='Frame' />
            <ExtractorCmd Cmd='LoadOne' />
            <ExtractorCmd Cmd='WriteLabel' Name='BigFrame' />
            <ExtractorCmd Cmd='GoTo' Bit='8' />
            <ExtractorCmd Cmd='ResetBitZero' />

            <Comment Value=' Making little frames of 4-bytes' />
            <ExtractorCmd Cmd='LoadRange' BitStart='0' BitEnd='7'/>
            <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                  BitTime='7' />
            <ExtractorCmd Cmd='LoadOne' />
            <ExtractorCmd Cmd='WriteLabel' Name='Frame' />
            <ExtractorCmd Cmd='LoadRange' BitStart='8' BitEnd='15'/>
            <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                  BitTime='15' />
            <ExtractorCmd Cmd='LoadRange' BitStart='16' BitEnd='23'/>
            <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                  BitTime='23' />
            <ExtractorCmd Cmd='LoadRange' BitStart='24' BitEnd='31'/>
            <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                  BitTime='31' />
```

```
                <ExtractorCmd Cmd='SubReg' Number='1' Value='4' />
                <ExtractorCmd Cmd='JumpCmpReg' Number='1' Value='4' />
                <ExtractorCmd Cmd='JumpForward' Amount='6' />
                <ExtractorCmd Cmd='JumpForward' Amount='1' />
                <ExtractorCmd Cmd='GoTo' Bit='32' />
                <ExtractorCmd Cmd='ResetBitZero' />
                <ExtractorCmd Cmd='JumpBackward' Amount='16' />
                <ExtractorCmd Cmd='WriteLabelTime' Name='Data'
                      BitTime='0' />
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
                <ExtractorCmd Cmd='JumpDone'/>
            </ExtractorCmds>
        </ExtractorSequence>
    </ExtractorSequences>
</ExtractorGrammar>
```

See Also

- Chapter 7, “Creating Extractor Algorithms,” starting on page 21
- “Using the FindPulseWidth Command" on page 31
- “Signal Extractor Commands" on page 52

Using the FindPulseWidth Command

The `FindPulseWidth` command has a complex algorithm that uses internal registers to calculate the smallest pulse width found on a specified input channel.

These registers must be set up before the `FindPulseWidth` command executes:

Register	Set Up Value
Reg 6	Min Range (used with Find range bit in Reg 10)
Reg 7	Max Range (used with Find range bit in Reg 10)
Reg 8	Set to zero
Reg 9	Set to zero
Reg 10	Find range(b2)/ Positive(b1)/ Negative(b0) Find pulses that match the above condition
Reg 11	Initial bit to start with
Reg 12	Increment of next bit to look at
Reg 13	Number of pulses to stop this search
Reg 14	Bit to stop this search
Reg 15	Initial Min pulse width

The `FindPulseWidth` command returns values in these registers:

Register	Return Value
Reg 15	Min pulse width found
Reg 11	Bit stopped at
Reg 9	Number of pulses found
Reg 8	Summation of pulses found (when Find range bit in Reg 10 is set)

The `FindPulseWidth` command stops searching after one of these conditions:

- The specified number of bits are evaluated (Reg 14).
- The specified number of pulses are found (Reg 13).
- The end of memory is found.
- The Signal Extractor **End Sample** is found.

In any case, the bit stopped at is returned in Reg 11 and the number of pulses found is returned in Reg9. When the search completes, the bit position is reset back to position 0. If the search hits the end of memory, the Signal Extractor algorithm is not terminated. (Typically, all other commands terminate the Signal Extractor algorithm if they hit the end of memory.)

The `FindPulseWidth` command is primarily intended to be used with normal timing data. It can be very helpful to find the minimum and average pulse width.

Reg10 specifies to look for positive pulses, negative pulses, or both. A value of 3 = measure both positive and negative pulses. A value of 2 = measure only positive pulses. A value of 1 = measure only negative pulses.

If the Find range bit (b2) of Reg 10 is set, the algorithm only finds pulses of the width specified between a minimum of Reg 6 and a maximum of Reg 7. In addition, all of the pulses found are summed into Reg 8. If you take Reg 8 and divide it by Reg 9, you get an integer average of the pulse width.

Reg11 specifies the initial bit to start with, and Reg 12 specifies the increment. If you have a single bit bus, you may want to start by finding a transition from 0 to 1 and then start on bit 1 with an increment of 1. If you have a multiple bit bus, you will want to set the increment to 2 or the appropriate value so that you are always looking at the same channel. As an example, if the bus is two bits wide and the bit you are interested in is bit 1, then you may want to first look for the pattern X0X1 and set Reg 11 = 3 (start on bit 3) and Reg 12 = 2 (Increment to every second bit).

Examples: extract RS-232 data

Here are two extractor algorithms to extract RS-232 data with the assumption that 1 start bit is high, 8 bits of data, 1 bit of parity, and 1 stop bit is low. The second version gets an integer average of the pulse width.

These algorithms use many of the recently added commands.

```
<ExtractorGrammar AlgorithmDescription='Extract RS232 data'
    DebugMode='On' >
  <Comment Value='

    This simple demo shows how you can extract RS-232 data.

  '/>
  <ExtractorLabels>
    <ExtractorLabel Name='RS232Data' Width='9' DefaultBase='Hex'/>
    <ExtractorLabel Name='Start' Width='1' DefaultBase='Hex'/>
    <ExtractorLabel Name='Stop' Width='1' DefaultBase='Hex'/>
    <ExtractorLabel Name='Error' Width=8' DefaultBase='Hex'/>
  </ExtractorLabels>
  <ExtractorSequences>
    <ExtractorSequence>
      <ExtractorPatterns>
        <ExtractorPattern Value='b01' Width='2' Enabled='T' />
      </ExtractorPatterns>
      <ExtractorCmds>
        <ExtractorCmd Cmd='JumpCmpReg' Number='10' Value='0'
                      Comment='First time through Reg 10 = 0' />
        <ExtractorCmd Cmd='JumpForward' Amount='3'
                      Comment='Not possible' />
        <ExtractorCmd Cmd='JumpForward' Amount='2'
                      Comment='Yes, do pulse width search' />
        <ExtractorCmd Cmd='JumpForward' Amount='21'
                      Comment='No, skip pulse width search' />

        <ExtractorCmd Cmd='MovReg' Number='10' Value='h3'
                      Comment='Find positive and negative pulses' />
        <ExtractorCmd Cmd='MovReg' Number='11' Value='1'
                      Comment='Bit to start with' />
        <ExtractorCmd Cmd='MovReg' Number='12' Value='1'
                      Comment='Increment' />
        <ExtractorCmd Cmd='MovReg' Number='13' Value='40'
                      Comment='Stop transitions' />
        <ExtractorCmd Cmd='MovReg' Number='14' Value='1000000'
                      Comment='Stop position' />
        <ExtractorCmd Cmd='MovReg' Number='15' Value='1000001'
                      Comment='Last Size' />
```

```
<ExtractorCmd Cmd='FindPulseWidth'
              Comment='Need to possibly loop' />

<ExtractorCmd Cmd='JumpCmpReg' Number='9' Value='30'
              Comment='Did we get at least 30 transitions?' />
<ExtractorCmd Cmd='JumpForward' Amount='6'
              Comment='No' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
              Comment='Yes' />
<ExtractorCmd Cmd='JumpCmpReg' Number='15' Value='5'
              Comment='Did we get at least 5 oversampling?' />
<ExtractorCmd Cmd='JumpForward' Amount='6'
              Comment='No' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
              Comment='Yes' />

<ExtractorCmd Cmd='JumpForward' Amount='7'
              Comment='Continue to look for RS232' />
<ExtractorCmd Cmd='EnablePattern' Number='1'
              Comment='Not enough transitions, do error' />
<ExtractorCmd Cmd='DisablePattern' Number='0' />
<ExtractorCmd Cmd='JumpDone'
              Comment='Enable Error Pattern' />
<ExtractorCmd Cmd='EnablePattern' Number='2'
              Comment='Not enough oversampling, do error' />
<ExtractorCmd Cmd='DisablePattern' Number='0' />
<ExtractorCmd Cmd='JumpDone'
              Comment='Enable Error Pattern' />


<Comment Value='Start of the data extraction' />
<ExtractorCmd Cmd='Mov2Regs' Number='14' Second='15'
              Comment='Save Width to Reg 14' />
<ExtractorCmd Cmd='DivReg' Number='14' Value='2'
              Comment='Divide Reg 14 by 2' />
<ExtractorCmd Cmd='AddReg' Number='14' Value='1'
          Comment='Initial point needs to be forward 1 bit' />


<ExtractorCmd Cmd='LoadZero' />
<ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
              Number='14'
              Comment='Write out value' />
<ExtractorCmd Cmd='LoadOne' />
<ExtractorCmd Cmd='WriteLabel' Name='Start'
              Comment='Write out value, Stop =0' />


<ExtractorCmd Cmd='MovReg' Number='1' Value='0'
              Comment='Reg 1 = count 8 bits' />
<ExtractorCmd Cmd='Mov2Regs' Number='2' Second=14'
              Comment='Reg 2 = Keeps the starting location' />


<ExtractorCmd Cmd='Add2Regs' Number='14' Second='15'
              Comment='Reset Reg 14' />
```

```
<ExtractorCmd Cmd='Mov2Regs' Number='12' Second='14'
              Comment='Set up for JumpCase3' />
<ExtractorCmd Cmd='SubReg' Number='12' Value='2'
              Comment='Set up for JumpCase3' />
<ExtractorCmd Cmd='Mov2Regs' Number='13' Second='14'
              Comment='Set up for JumpCase3' />
<ExtractorCmd Cmd='SubReg' Number='13' Value='1'
              Comment='Set up for JumpCase3' />

<ExtractorCmd Cmd='JumpCase3BitReg' Reg1='12' Reg2='13'
              Reg3='14'
              Comment='Test for transition' />
<ExtractorCmd Cmd='JumpForward' Amount='10'
              Comment='000 Do nothing' />
<ExtractorCmd Cmd='JumpForward' Amount='7'
              Comment='001 Push by 2' />
<ExtractorCmd Cmd='JumpForward' Amount='8'
              Comment='010 Not possible' />
<ExtractorCmd Cmd='JumpForward' Amount='6'
              Comment='011 Push by 1' />
<ExtractorCmd Cmd='JumpForward' Amount='5'
              Comment='100 Push by 1' />
<ExtractorCmd Cmd='JumpForward' Amount='5'
              Comment='101 Not possible' />
<ExtractorCmd Cmd='JumpForward' Amount='2'
              Comment='110 Push by 2' />
<ExtractorCmd Cmd='JumpForward' Amount='3'
              Comment='111 Do nothing' />
<ExtractorCmd Cmd='AddReg' Number='14' Value='1'
              Comment='Push 1 bit' />
<ExtractorCmd Cmd='AddReg' Number='14' Value='1'
              Comment='Push 1 bit' />

<ExtractorCmd Cmd='LoadBitReg' Number='14'
              Comment='Load this bit' />
<ExtractorCmd Cmd='AddReg' Number='1' Value='1'
              Comment='Counting total bits' />
<ExtractorCmd Cmd='JumpCmpReg' Number='1' Value='9'
              Comment='Did we get 9 bits?' />
<ExtractorCmd Cmd='JumpBackward' Amount='19'
              Comment='Go to next bit' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
              Comment='Last bit found' />


<ExtractorCmd Cmd='Add2Regs' Number='2' Second=15'
              Comment='Going to bit position of start'/>

<ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
              Number='2'
              Comment='Write out value, Start/Stop=0'/>

<ExtractorCmd Cmd='Add2Regs' Number='14' Second='15'
              Comment='Position bit Stop' />
<ExtractorCmd Cmd='GoToReg' Number='14'
              Comment='Go to Bit 0' />
```

```
    <ExtractorCmd Cmd='LoadZero' />
    <ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
                  Number='14'
                  Comment='Write out value' />
    <ExtractorCmd Cmd='LoadOne' />
    <ExtractorCmd Cmd='WriteLabel' Name='Stop'
                  Comment='Write out value' />
    <ExtractorCmd Cmd='JumpCase1BitReg' Reg1='14'
                  Comment='Test legal stop' />
    <ExtractorCmd Cmd='JumpDone'
                  Comment='Legal stop bit of zero' />
    <ExtractorCmd Cmd='LoadOne' />
    <ExtractorCmd Cmd='WriteLabel' Name='Error'
                  Comment='Write out value' />
    <ExtractorCmd Cmd='EnablePattern' Number='3'
              Comment='Skipping one transition, try to align' />
    <ExtractorCmd Cmd='DisablePattern' Number='0' />
    <ExtractorCmd Cmd='JumpDone' />

  </ExtractorCmds>
</ExtractorSequence>

<ExtractorSequence>
  <ExtractorPatterns>
    <ExtractorPattern Value='b01' Width='2' Enabled='F'
                      Comment='Error not enough transitions' />
  </ExtractorPatterns>
  <ExtractorCmds>
    <ExtractorCmd Cmd='LoadZero'
                  Comment='Load Zero' />
    <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                  Comment='Write out value as zero' />
    <ExtractorCmd Cmd='MovReg' Number='0' Value='h2'
                  Comment='Not enough transitions' />
    <ExtractorCmd Cmd='WriteLabel' Name='Error'
                  Comment='Write out Error value' />
    <ExtractorCmd Cmd='JumpDone' />
  </ExtractorCmds>
</ExtractorSequence>

<ExtractorSequence>
  <ExtractorPatterns>
    <ExtractorPattern Value='b01' Width='2' Enabled='F'
                      Comment='Error not enough oversampling' />
  </ExtractorPatterns>
  <ExtractorCmds>
    <ExtractorCmd Cmd='LoadZero'
                  Comment='Load Zero' />
    <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                  Comment='Write out value as zero' />
    <ExtractorCmd Cmd='MovReg' Number='0' Value='h4'
                  Comment='Not enough oversampling' />
    <ExtractorCmd Cmd='WriteLabel' Name='Error'
                  Comment='Write out Error value' />
    <ExtractorCmd Cmd='JumpDone' />
  </ExtractorCmds>
</ExtractorSequence>
```

```
      <ExtractorSequence>
       <ExtractorPatterns>
         <ExtractorPattern Value='b01' Width='2' Enabled='F'
                       Comment='Error wrong stop bit, skip one edge' />
       </ExtractorPatterns>
       <ExtractorCmds>
         <ExtractorCmd Cmd='LoadZero'
                       Comment='Load Zero' />
         <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                       Comment='Write out value as zero' />
         <ExtractorCmd Cmd='MovReg' Number='0' Value='h8'
                       Comment='Not enough oversampling' />
         <ExtractorCmd Cmd='WriteLabel' Name='Error'
                       Comment='Write out Error value' />
         <ExtractorCmd Cmd='EnablePattern' Number='0'
                       Comment='Try to align' />
         <ExtractorCmd Cmd='DisablePattern' Number='3'/>
         <ExtractorCmd Cmd='JumpDone' />
       </ExtractorCmds>
     </ExtractorSequence>

  </ExtractorSequences>
</ExtractorGrammar>
```

The second RS-232 data Signal Extractor algorithm:

```
<ExtractorGrammar AlgorithmDescription='Extract RS232 data'
    DebugMode='On' >
  <Comment Value='

    This simple demo shows how you can extract RS-232 data.

  '/>
  <ExtractorLabels>
    <ExtractorLabel Name='RS232Data' Width='9' DefaultBase='Hex'/>
    <ExtractorLabel Name='Start' Width='1' DefaultBase='Hex'/>
    <ExtractorLabel Name='Stop' Width='1' DefaultBase='Hex'/>
    <ExtractorLabel Name='Error' Width=8' DefaultBase='Hex'/>
  </ExtractorLabels>
  <ExtractorSequences>
    <ExtractorSequence>
      <ExtractorPatterns>
        <ExtractorPattern Value='b01' Width='2' Enabled='T' />
      </ExtractorPatterns>
      <ExtractorCmds>
        <ExtractorCmd Cmd='JumpCmpReg' Number='10' Value='0'
                      Comment='First time through Reg 10 = 0' />
        <ExtractorCmd Cmd='JumpForward' Amount='3'
                      Comment='Not possible' />
        <ExtractorCmd Cmd='JumpForward' Amount='2'
                      Comment='Yes, do pulse width search' />
        <ExtractorCmd Cmd='JumpForward' Amount='32'
                      Comment='No, skip pulse width search' />

        <ExtractorCmd Cmd='MovReg' Number='10' Value='h3'
                      Comment='Find positive and negative pulses' />
```

```
<ExtractorCmd Cmd='MovReg' Number='11' Value='1'
              Comment='Bit to start with' />
<ExtractorCmd Cmd='MovReg' Number='12' Value='1'
              Comment='Increment' />
<ExtractorCmd Cmd='MovReg' Number='13' Value='40'
              Comment='Stop transitions' />
<ExtractorCmd Cmd='MovReg' Number='14' Value='1000000'
              Comment='Stop position' />
<ExtractorCmd Cmd='MovReg' Number='15' Value='1000001'
              Comment='Last Size' />
<ExtractorCmd Cmd='FindPulseWidth'
              Comment='Need to possibly loop' />

<ExtractorCmd Cmd='JumpCmpReg' Number='9' Value='30'
              Comment='Did we get at least 30 transitions?' />
<ExtractorCmd Cmd='JumpForward' Amount='6'
              Comment='No' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
              Comment='Yes' />
<ExtractorCmd Cmd='JumpCmpReg' Number='15' Value='5'
              Comment='Did we get at least 5 oversampling?' />
<ExtractorCmd Cmd='JumpForward' Amount='6'
              Comment='No' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
              Comment='Yes' />

<ExtractorCmd Cmd='JumpForward' Amount='7'
              Comment='Continue to look for RS232' />
<ExtractorCmd Cmd='EnablePattern' Number='1'
              Comment='Not enough transitions, do error' />
<ExtractorCmd Cmd='DisablePattern' Number='0' />
<ExtractorCmd Cmd='JumpDone'
              Comment='Enable Error Pattern' />
<ExtractorCmd Cmd='EnablePattern' Number='2'
              Comment='Not enough oversampling, do error' />
<ExtractorCmd Cmd='DisablePattern' Number='0' />
<ExtractorCmd Cmd='JumpDone'
              Comment='Enable Error Pattern' />


<Comment Value='Find the width average value, then multiply
                by 10' />
<ExtractorCmd Cmd='Mov2Regs' Number='14' Second='11'
              Comment='Use Stop bit position from last find' />
<ExtractorCmd Cmd='MovReg' Number='9' Value='h0'
              Comment='Reset the pulses found to 0' />
<ExtractorCmd Cmd='MovReg' Number='10' Value='h7'
         Comment='Find range, positive and negative pulses' />
<ExtractorCmd Cmd='MovReg' Number='11' Value='1'
              Comment='Bit to start with' />
<ExtractorCmd Cmd='Mov2Regs' Number='6' Second='15'
              Comment='Min Value' />
<ExtractorCmd Cmd='Mov2Regs' Number='7' Second='15'
              Comment='Max Value' />
<ExtractorCmd Cmd='AddReg' Number='7' Value='2'
              Comment='Max Value = MinValue+2' />
<ExtractorCmd Cmd='FindPulseWidth'
```

```
                Comment='Find range Min to Min+2' />

<ExtractorCmd Cmd='MultReg' Number='8' Value='100'
          Comment='Multiply the summation of pulses by 100'/>
<ExtractorCmd Cmd='Div2Regs' Number='8' Second='9'
    Comment='Divide the sumation of pulses by # pulses found'/>
<ExtractorCmd Cmd='Mov2Regs' Number='15' Second='8'
                Comment='Reg 15 is pulse width * 100'/>


<Comment Value='Start of the data extraction' />
<ExtractorCmd Cmd='Mov2Regs' Number='14' Second='15'
                Comment='Save Width to Reg 14'/>
<ExtractorCmd Cmd='DivReg' Number='14' Value='2'
                Comment='Divide Reg 14 pulse width*100 by 2
                         = 1/2 pulse width '/>
<ExtractorCmd Cmd='AddReg' Number='14' Value='100'
          Comment='Initial point needs to be forward 1 bit'/>
<ExtractorCmd Cmd='Mov2Regs' Number='13' Second='14'
                Comment='Save Width to Reg 13'/>

<ExtractorCmd Cmd='DivReg' Number='13' Value='100'
                Comment='Divide Reg 13 pulse width*100 by 100' />
<ExtractorCmd Cmd='LoadZero' />
<ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
                Number='13'
                Comment='Write out value' />
<ExtractorCmd Cmd='LoadOne' />
<ExtractorCmd Cmd='WriteLabel' Name='Start'
                Comment='Write out value, Stop =0' />


<ExtractorCmd Cmd='MovReg' Number='1' Value='0'
                Comment='Reg 1 = count 8 bits'/>
<ExtractorCmd Cmd='Mov2Regs' Number='2' Second=14'
                Comment='Reg 2 = Keeps the starting location' />


<ExtractorCmd Cmd='Add2Regs' Number='14' Second='15'
                Comment='Reset Reg 14'/>

<ExtractorCmd Cmd='Mov2Regs' Number='13' Second='14'
                Comment='Set up for next bit'/>
<ExtractorCmd Cmd='DivReg' Number='13' Value='100'
                Comment='Set up for actual bit location'/>

<ExtractorCmd Cmd='LoadBitReg' Number='13'
                Comment='Load this bit' />
<ExtractorCmd Cmd='AddReg' Number='1' Value='1'
                Comment='Counting total bits' />
<ExtractorCmd Cmd='JumpCmpReg' Number='1' Value='9'
                Comment='Did we get 9 bits?' />
<ExtractorCmd Cmd='JumpBackward' Amount='6'
                Comment='Go to next bit' />
<ExtractorCmd Cmd='JumpForward' Amount='1'
                Comment='Last bit found' />
```

```
    <ExtractorCmd Cmd='Add2Regs' Number='2' Second=15'
                  Comment='Going to bit position of start' />
    <ExtractorCmd Cmd='DivReg' Number='2' Value='100'
                  Comment='Set up for actual bit location' />
    <ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
                  Number='2'
                  Comment='Write out value, Start/Stop=0' />

    <ExtractorCmd Cmd='Add2Regs' Number='14' Second='15'
                  Comment='Position bit Stop' />
    <ExtractorCmd Cmd='DivReg' Number='14' Value='100'
                  Comment='Set up for actual bit location' />
    <ExtractorCmd Cmd='GoToReg' Number='14'
                  Comment='Go to Bit 0' />

    <ExtractorCmd Cmd='LoadZero' />
    <ExtractorCmd Cmd='WriteLabelTimeReg' Name='RS232Data'
                  Number='14'
                  Comment='Write out value' />
    <ExtractorCmd Cmd='LoadOne' />
    <ExtractorCmd Cmd='WriteLabel' Name='Stop'
                  Comment='Write out value' />
    <ExtractorCmd Cmd='JumpCase1BitReg' Reg1='14'
                  Comment='Test legal stop' />
    <ExtractorCmd Cmd='JumpDone'
                  Comment='Legal stop bit of zero'/>
    <ExtractorCmd Cmd='LoadOne' />
    <ExtractorCmd Cmd='WriteLabel' Name='Error'
                  Comment='Write out value'/>
    <ExtractorCmd Cmd='EnablePattern' Number='3'
                  Comment='Skipping one transition, try to align' />
    <ExtractorCmd Cmd='DisablePattern' Number='0' />
    <ExtractorCmd Cmd='JumpDone' />

  </ExtractorCmds>
</ExtractorSequence>

<ExtractorSequence>
  <ExtractorPatterns>
    <ExtractorPattern Value='b01' Width='2' Enabled='F'
                      Comment='Error not enough transitions' />
  </ExtractorPatterns>
  <ExtractorCmds>
    <ExtractorCmd Cmd='LoadZero'
                  Comment='Load Zero' />
    <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                  Comment='Write out value as zero' />
    <ExtractorCmd Cmd='MovReg' Number='0' Value='h2'
                  Comment='Not enough transitions' />
    <ExtractorCmd Cmd='WriteLabel' Name='Error'
                  Comment='Write out Error value' />
    <ExtractorCmd Cmd='JumpDone' />
  </ExtractorCmds>
</ExtractorSequence>

<ExtractorSequence>
```

```
      <ExtractorPatterns>
        <ExtractorPattern Value='b01' Width='2' Enabled='F'
                          Comment='Error not enough oversampling' />
      </ExtractorPatterns>
      <ExtractorCmds>
        <ExtractorCmd Cmd='LoadZero'
                      Comment='Load Zero' />
        <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                      Comment='Write out value as zero' />
        <ExtractorCmd Cmd='MovReg' Number='0' Value='h4'
                      Comment='Not enough oversampling' />
        <ExtractorCmd Cmd='WriteLabel' Name='Error'
                      Comment='Write out Error value' />
        <ExtractorCmd Cmd='JumpDone' />
      </ExtractorCmds>
    </ExtractorSequence>

    <ExtractorSequence>
      <ExtractorPatterns>
        <ExtractorPattern Value='b01' Width='2' Enabled='F'
                        Comment='Error wrong stop bit, skip one edge' />
      </ExtractorPatterns>
      <ExtractorCmds>
        <ExtractorCmd Cmd='LoadZero'
                      Comment='Load Zero' />
        <ExtractorCmd Cmd='WriteLabelTime' Name='RS232Data' BitTime='1'
                      Comment='Write out value as zero' />
        <ExtractorCmd Cmd='MovReg' Number='0' Value='h8'
                      Comment='Not enough oversampling' />
        <ExtractorCmd Cmd='WriteLabel' Name='Error'
                      Comment='Write out Error value' />
        <ExtractorCmd Cmd='EnablePattern' Number='0'
                      Comment='Try to align' />
        <ExtractorCmd Cmd='DisablePattern' Number='3'/>
        <ExtractorCmd Cmd='JumpDone' />
      </ExtractorCmds>
    </ExtractorSequence>

  </ExtractorSequences>
</ExtractorGrammar>
```

See Also

- Chapter 7, “Creating Extractor Algorithms,” starting on page 21
- “Using Register Commands" on page 27
- “Signal Extractor Commands" on page 52

Using Debug Mode

You can debug Signal Extractor algorithms by turning on debug mode:

1 Use the DebugMode='On' attribute in the `<ExtractorGrammar>` (see page 46) element. For example:

```
<ExtractorGrammar AlgorithmDescription='Some description'
      DebugMode='On' >
```

2 In the *Keysight Logic Analyzer* application, choose **Edit->Options...**.

3 In the Options dialog, check **Event Logging**.

When you complete debugging, you will want to uncheck the Event Logging.

4 Load the Signal Extractor with the algorithm file that you want to debug. See Chapter 6, “Adding a Signal Extractor Tool,” starting on page 19.

5 Now, when the Signal Extractor is used, the debug output will be sent to the logic analyzer event logs which are located in \My Documents\Keysight Technologies\Logic Analyzer\Log Files. Examine the agLastLogFolder.txt file to determine the folder to look at.

Example Debug Output

```
d ??? IPat=00000000000000000000000000000008
Mask=00000000000000000000000000000001
Test=00000000000000000000000000000001
d ??? IPat=00000000000000000000000000000809
Mask=00000000000000000000000000000001
Test=00000000000000000000000000000001
d ??? CMD=LOADRANGE Bit_Start=0, Bit_End=7
Reg0=00000000000000000000000000000009
d ??? CMD=Write_Time Label=0 Bit_Time=0
Reg0=00000000000000000000000000000009
d ??? CMD=JUMPDONE
```

Extractor Algorithm XML Format

You can create Signal Extractor algorithms as XML format files with the following XML elements and hierarchy:

```
<ExtractorGrammar> (see page 46)
   <ExtractorLabels> (see page 48)
      <ExtractorLabel> (see page 47)
      <ExtractorFolder> (see page 46)
         <ExtractorLabel> (see page 47)
   <ExtractorSequences> (see page 50)
      <ExtractorSequence> (see page 49)
         <ExtractorPatterns> (see page 49)
            <ExtractorPattern> (see page 48)
         <ExtractorCmds> (see page 45)
            <ExtractorCmd> (see page 43)
```

Example Note that you can have multiple `<ExtractorSequence>` elements to define different sets of commands for different patterns.

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
         </ExtractorPatterns>
         <ExtractorCmds>
            <ExtractorCmd Cmd='Load' Bit='2' />
            <ExtractorCmd Cmd='Load' Bit='5' />
            <ExtractorCmd Cmd='Load' Bit='8' />
            <ExtractorCmd Cmd='Load' Bit='11' />
            <ExtractorCmd Cmd='Load' Bit='14' />
            <ExtractorCmd Cmd='Load' Bit='17' />
            <ExtractorCmd Cmd='Load' Bit='20' />
            <ExtractorCmd Cmd='Load' Bit='23' />
            <ExtractorCmd Cmd='Load' Bit='26' />
            <ExtractorCmd Cmd='Load' Bit='29' />
            <ExtractorCmd Cmd='Load' Bit='32' />
            <ExtractorCmd Cmd='Load' Bit='35' />
            <ExtractorCmd Cmd='Load' Bit='38' />
            <ExtractorCmd Cmd='Load' Bit='41' />
            <ExtractorCmd Cmd='Load' Bit='44' />
            <ExtractorCmd Cmd='Load' Bit='47' />
            <ExtractorCmd Cmd='WriteLabelTime' Name='Idata'
                  BitTime='0' />
            <ExtractorCmd Cmd='Load' Bit='1' />
            <ExtractorCmd Cmd='Load' Bit='4' />
```

```
            <ExtractorCmd Cmd='Load' Bit='7' />
            <ExtractorCmd Cmd='Load' Bit='10' />
            <ExtractorCmd Cmd='Load' Bit='13' />
            <ExtractorCmd Cmd='Load' Bit='16' />
            <ExtractorCmd Cmd='Load' Bit='19' />
            <ExtractorCmd Cmd='Load' Bit='22' />
            <ExtractorCmd Cmd='Load' Bit='25' />
            <ExtractorCmd Cmd='Load' Bit='28' />
            <ExtractorCmd Cmd='Load' Bit='31' />
            <ExtractorCmd Cmd='Load' Bit='34' />
            <ExtractorCmd Cmd='Load' Bit='37' />
            <ExtractorCmd Cmd='Load' Bit='40' />
            <ExtractorCmd Cmd='Load' Bit='43' />
            <ExtractorCmd Cmd='Load' Bit='46' />
            <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
            <ExtractorCmd Cmd='JumpDone' />
         </ExtractorCmds>
      </ExtractorSequence>
   </ExtractorSequences>
</ExtractorGrammar>
```

See Also

- Chapter 7, “Creating Extractor Algorithms,” starting on page 21

<ExtractorCmd> Element

`<ExtractorCmd>` elements are the heart of the Signal Extractor tool. They let you extract bits into a value and then write the value to a bus/signal.

Attributes

Name	Description
`Amount`	`'number'` – Used with `Cmd='JumpBackward'`, `Cmd='JumpForward'`, `Cmd='Split'`.
`Bit`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='GoTo'`, `Cmd='Load'`.
`Bit1`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='JumpCase1Bit'`, `Cmd='JumpCase2Bit'`, `Cmd='JumpCase3Bit'`, `Cmd='JumpCase4Bit'`.
`Bit2`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='JumpCase2Bit'`, `Cmd='JumpCase3Bit'`, `Cmd='JumpCase4Bit'`.
`Bit3`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='JumpCase3Bit'`, `Cmd='JumpCase4Bit'`.
`Bit4`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='JumpCase4Bit'`.
`BitEnd`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='LoadRange'`.
`BitStart`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='LoadRange'`.
`BitTime`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='WriteLabelTime'`.
`BitTimeEnd`	`'number'` – Non-negative signed 32-bit integer, used with `Cmd='WriteLabelTimeDelta'`.

BitTimeStart	'number' – Non-negative signed 32-bit integer, used with Cmd='WriteLabelTimeDelta'.
Cmd	'string' – Name of the command, used with all commands. See "Signal Extractor Commands" on page 52.
Limit	'number' – From 1 to 32, used with signed limit register addition commands.
Name	'string' – Name of a bus/signal, used with Cmd='WriteLabel', Cmd='WriteLabelTime', Cmd='WriteLabelTimeDelta', Cmd='Split'.
Number	'zero-based_number' – Used with Cmd='DisablePattern', Cmd='EnablePattern'.
Second	'zero-based_number' – Used with register commands to identify the second register (0, 1, 2, or 3).
Size	'number' – Used with Cmd='Split'.
TimeDen	'decimal' – Denominator in fraction, used with Cmd='WriteLabelTimeDelta'.
TimeNum	'decimal' – Numerator in fraction, used with Cmd='WriteLabelTimeDelta'.
TimePS	'number' – Time in picoseconds, used with Cmd='JumpTimeGreaterEqual'.
Value	'number' – 32-bit integer, used with register commands.

Parents This element can have the following parents: <ExtractorCmds> (see page 45).

Example

```
<ExtractorCmds>
   <ExtractorCmd Cmd='Load' Bit='2' />
   <ExtractorCmd Cmd='Load' Bit='5' />
   <ExtractorCmd Cmd='Load' Bit='8' />
   <ExtractorCmd Cmd='Load' Bit='11' />
   <ExtractorCmd Cmd='Load' Bit='14' />
   <ExtractorCmd Cmd='Load' Bit='17' />
   <ExtractorCmd Cmd='Load' Bit='20' />
   <ExtractorCmd Cmd='Load' Bit='23' />
   <ExtractorCmd Cmd='Load' Bit='26' />
   <ExtractorCmd Cmd='Load' Bit='29' />
   <ExtractorCmd Cmd='Load' Bit='32' />
   <ExtractorCmd Cmd='Load' Bit='35' />
   <ExtractorCmd Cmd='Load' Bit='38' />
   <ExtractorCmd Cmd='Load' Bit='41' />
   <ExtractorCmd Cmd='Load' Bit='44' />
   <ExtractorCmd Cmd='Load' Bit='47' />
   <ExtractorCmd Cmd='WriteLabelTime' Name='Idata' BitTime='0' />
   <ExtractorCmd Cmd='Load' Bit='1' />
   <ExtractorCmd Cmd='Load' Bit='4' />
   <ExtractorCmd Cmd='Load' Bit='7' />
   <ExtractorCmd Cmd='Load' Bit='10' />
   <ExtractorCmd Cmd='Load' Bit='13' />
   <ExtractorCmd Cmd='Load' Bit='16' />
   <ExtractorCmd Cmd='Load' Bit='19' />
   <ExtractorCmd Cmd='Load' Bit='22' />
```

```
   <ExtractorCmd Cmd='Load' Bit='25' />
   <ExtractorCmd Cmd='Load' Bit='28' />
   <ExtractorCmd Cmd='Load' Bit='31' />
   <ExtractorCmd Cmd='Load' Bit='34' />
   <ExtractorCmd Cmd='Load' Bit='37' />
   <ExtractorCmd Cmd='Load' Bit='40' />
   <ExtractorCmd Cmd='Load' Bit='43' />
   <ExtractorCmd Cmd='Load' Bit='46' />
   <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
   <ExtractorCmd Cmd='JumpDone' />
</ExtractorCmds>
```

<ExtractorCmds> Element

The `<ExtractorCmds>` element contains `<ExtractorCmd>` elements.

Children This element can have the following children: `<ExtractorCmd>` (see page 43).

Parents This element can have the following parents: `<ExtractorSequence>` (see page 49).

Example

```
<ExtractorCmds>
   <ExtractorCmd Cmd='Load' Bit='2' />
   <ExtractorCmd Cmd='Load' Bit='5' />
   <ExtractorCmd Cmd='Load' Bit='8' />
   <ExtractorCmd Cmd='Load' Bit='11' />
   <ExtractorCmd Cmd='Load' Bit='14' />
   <ExtractorCmd Cmd='Load' Bit='17' />
   <ExtractorCmd Cmd='Load' Bit='20' />
   <ExtractorCmd Cmd='Load' Bit='23' />
   <ExtractorCmd Cmd='Load' Bit='26' />
   <ExtractorCmd Cmd='Load' Bit='29' />
   <ExtractorCmd Cmd='Load' Bit='32' />
   <ExtractorCmd Cmd='Load' Bit='35' />
   <ExtractorCmd Cmd='Load' Bit='38' />
   <ExtractorCmd Cmd='Load' Bit='41' />
   <ExtractorCmd Cmd='Load' Bit='44' />
   <ExtractorCmd Cmd='Load' Bit='47' />
   <ExtractorCmd Cmd='WriteLabelTime' Name='Idata' BitTime='0' />
   <ExtractorCmd Cmd='Load' Bit='1' />
   <ExtractorCmd Cmd='Load' Bit='4' />
   <ExtractorCmd Cmd='Load' Bit='7' />
   <ExtractorCmd Cmd='Load' Bit='10' />
   <ExtractorCmd Cmd='Load' Bit='13' />
   <ExtractorCmd Cmd='Load' Bit='16' />
   <ExtractorCmd Cmd='Load' Bit='19' />
   <ExtractorCmd Cmd='Load' Bit='22' />
   <ExtractorCmd Cmd='Load' Bit='25' />
   <ExtractorCmd Cmd='Load' Bit='28' />
   <ExtractorCmd Cmd='Load' Bit='31' />
   <ExtractorCmd Cmd='Load' Bit='34' />
   <ExtractorCmd Cmd='Load' Bit='37' />
   <ExtractorCmd Cmd='Load' Bit='40' />
   <ExtractorCmd Cmd='Load' Bit='43' />
   <ExtractorCmd Cmd='Load' Bit='46' />
   <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
   <ExtractorCmd Cmd='JumpDone' />
</ExtractorCmds>
```

<ExtractorFolder> Element

The <ExtractorFolder> element contains output bus/signal definitions for a second time base.

Attributes

Name	Description
FolderName	'string' – Name for second group of 4 bus/signals. This is only needed if you are going to create a second time base and group of up to 4 bus/signals.

Children This element can have the following children: <ExtractorLabel> (see page 47).

Parents This element can have the following parents: <ExtractorLabels> (see page 48).

Example

```
<ExtractorFolder FolderName='SecondBus'>
   <ExtractorLabel Name='FrameMarker' Width='1' DefaultBase='Binary' />
</ExtractorFolder>
```

<ExtractorGrammar> Element

The <ExtractorGrammar> element is the top element in an XML format Signal Extractor algorithm file.

Attributes

Name	Description
AlgorithmDescription	'string' – Short text description of the algorithm.
DebugMode	'On' – If present, this attribute causes all Signal Extractor operations to be sent to the logic analyzer event logging file. See "Using Debug Mode" on page 41.
InputMode	'Serialize' – If present, this attribute will cause the Signal Extractor tool to serialize the input bus/signal into a single bit stream before it starts searching for patterns.

Children This element can have the following children: <ExtractorLabels> (see page 48), <ExtractorSequences> (see page 50).

Parents None.

Example

```
<ExtractorGrammar AlgorithmDescription='IQ data from 3-bit Serial' >
   <ExtractorLabels>
      <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
            VSAOutput='T' VSACompressionFactor='-16'/>
      <ExtractorFolder FolderName='SecondBus'>
         <ExtractorLabel Name='FrameMarker' Width='1'
               DefaultBase='Binary' />
      </ExtractorFolder>
   </ExtractorLabels>
   <ExtractorSequences>
      <ExtractorSequence>
         <ExtractorPatterns>
            <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
         </ExtractorPatterns>
         <ExtractorCmds>
```

```
            <ExtractorCmd Cmd='Load' Bit='2' />
            <ExtractorCmd Cmd='Load' Bit='5' />
            <ExtractorCmd Cmd='Load' Bit='8' />
            <ExtractorCmd Cmd='Load' Bit='11' />
            <ExtractorCmd Cmd='Load' Bit='14' />
            <ExtractorCmd Cmd='Load' Bit='17' />
            <ExtractorCmd Cmd='Load' Bit='20' />
            <ExtractorCmd Cmd='Load' Bit='23' />
            <ExtractorCmd Cmd='Load' Bit='26' />
            <ExtractorCmd Cmd='Load' Bit='29' />
            <ExtractorCmd Cmd='Load' Bit='32' />
            <ExtractorCmd Cmd='Load' Bit='35' />
            <ExtractorCmd Cmd='Load' Bit='38' />
            <ExtractorCmd Cmd='Load' Bit='41' />
            <ExtractorCmd Cmd='Load' Bit='44' />
            <ExtractorCmd Cmd='Load' Bit='47' />
            <ExtractorCmd Cmd='WriteLabelTime' Name='Idata'
                  BitTime='0' />
            <ExtractorCmd Cmd='Load' Bit='1' />
            <ExtractorCmd Cmd='Load' Bit='4' />
            <ExtractorCmd Cmd='Load' Bit='7' />
            <ExtractorCmd Cmd='Load' Bit='10' />
            <ExtractorCmd Cmd='Load' Bit='13' />
            <ExtractorCmd Cmd='Load' Bit='16' />
            <ExtractorCmd Cmd='Load' Bit='19' />
            <ExtractorCmd Cmd='Load' Bit='22' />
            <ExtractorCmd Cmd='Load' Bit='25' />
            <ExtractorCmd Cmd='Load' Bit='28' />
            <ExtractorCmd Cmd='Load' Bit='31' />
            <ExtractorCmd Cmd='Load' Bit='34' />
            <ExtractorCmd Cmd='Load' Bit='37' />
            <ExtractorCmd Cmd='Load' Bit='40' />
            <ExtractorCmd Cmd='Load' Bit='43' />
            <ExtractorCmd Cmd='Load' Bit='46' />
            <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
            <ExtractorCmd Cmd='JumpDone' />
         </ExtractorCmds>
      </ExtractorSequence>
   </ExtractorSequences>
</ExtractorGrammar>
```

<ExtractorLabel> Element

The `<ExtractorLabel>` element contains an output bus/signal definition.

Attributes

Name	Description
`DefaultBase`	`'Binary'`, `'Hex'`, `'Octal'`, `'Decimal'`, `'Signed Decimal'`
`Name`	`'string'` – Text name of output bus/signal to display.

VSACompressionFactor	'number' – Optional value used to tell the Keysight 89601A vector signal analysis software about the serial compression or expansion that is occurring between the input data to the output data. A value of -16 implies that for every 16 input states the output bus/signal has 1 state. A value of +4 implies that for every 1 input state we are generating 4 output states.
VSAOutput	'F' – The bus/signal cannot be used with the Keysight 89601A vector signal analysis software. 'T' – The bus/signal can be used with the Keysight 89601A vector signal analysis software.
Width	'number' – Width of the output bus/signal in channels (1 to 128).

Parents This element can have the following parents: <ExtractorLabels> (see page 48), <ExtractorFolder> (see page 46).

Example

```
<ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
      VSAOutput='T' VSACompressionFactor='-16'/>
```

<ExtractorLabels> Element

The <ExtractorLabels> element contains output bus/signal and bus/signal folder definitions.

Children This element can have the following children: <ExtractorLabel> (see page 47), <ExtractorFolder> (see page 46).

Parents This element can have the following parents: <ExtractorGrammar> (see page 46).

Example

```
<ExtractorLabels>
   <ExtractorLabel Name='Idata' Width='16' DefaultBase='Hex'
         VSAOutput='T' VSACompressionFactor='-16'/>
   <ExtractorLabel Name='Qdata' Width='16' DefaultBase='Hex'
         VSAOutput='T' VSACompressionFactor='-16'/>
   <ExtractorFolder FolderName='SecondBus'>
      <ExtractorLabel Name='FrameMarker' Width='1'
            DefaultBase='Binary' />
   </ExtractorFolder>
</ExtractorLabels>
```

<ExtractorPattern> Element

The <ExtractorPattern> element describes a pattern to be found.

Attributes

Name	Description
Enabled	'F' – To indicate that this pattern is initially disabled. 'T' – If this pattern is initially enabled.
Value	'binary_or_hex_number' – Binary or hex pattern to search for first. Use a leading 'b' or 'h' and 'X' for don't care on some bits.
Width	'number' – Width of the pattern that you are looking for (1 to 128).

Parents This element can have the following parents: <ExtractorPatterns> (see page 49).

Example

```
<ExtractorPatterns>
   <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
</ExtractorPatterns>
```

<ExtractorPatterns> Element

The `<ExtractorPatterns>` element contains `<ExtractorPattern>` elements.

Children This element can have the following children: `<ExtractorPattern>` (see page 48).

Parents This element can have the following parents: `<ExtractorSequence>` (see page 49).

Example

```
<ExtractorPatterns>
   <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
</ExtractorPatterns>
```

<ExtractorSequence> Element

Each extractor sequence consists of a set of extractor patterns and extractor commands. When one of the extractor patterns matches, it will start executing the corresponding extractor commands.

Multiple occurrences of `ExtractorSequence` can exist. Make sure the more specific `ExtractorSequence` occurs before the more generic `ExtractorSequence`. For example, you may have an `ExtractorSequence` to match for a particular type of serial frame and extract I and Q data from it as the first `ExtractorSequence`. Then, you may have a default `ExtractorSequence` to match all other types of serial frames that you are trying to determine sizes of so that you can skip past them properly.

Children This element can have the following children: `<ExtractorPatterns>` (see page 49), `<ExtractorCmds>` (see page 45).

Parents This element can have the following parents: `<ExtractorSequences>` (see page 50).

Example

```
<ExtractorSequence>
   <ExtractorPatterns>
      <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
   </ExtractorPatterns>
   <ExtractorCmds>
      <ExtractorCmd Cmd='Load' Bit='2' />
      <ExtractorCmd Cmd='Load' Bit='5' />
      <ExtractorCmd Cmd='Load' Bit='8' />
      <ExtractorCmd Cmd='Load' Bit='11' />
      <ExtractorCmd Cmd='Load' Bit='14' />
      <ExtractorCmd Cmd='Load' Bit='17' />
      <ExtractorCmd Cmd='Load' Bit='20' />
      <ExtractorCmd Cmd='Load' Bit='23' />
      <ExtractorCmd Cmd='Load' Bit='26' />
      <ExtractorCmd Cmd='Load' Bit='29' />
      <ExtractorCmd Cmd='Load' Bit='32' />
      <ExtractorCmd Cmd='Load' Bit='35' />
      <ExtractorCmd Cmd='Load' Bit='38' />
      <ExtractorCmd Cmd='Load' Bit='41' />
      <ExtractorCmd Cmd='Load' Bit='44' />
      <ExtractorCmd Cmd='Load' Bit='47' />
      <ExtractorCmd Cmd='WriteLabelTime' Name='Idata' BitTime='0' />
      <ExtractorCmd Cmd='Load' Bit='1' />
      <ExtractorCmd Cmd='Load' Bit='4' />
      <ExtractorCmd Cmd='Load' Bit='7' />
      <ExtractorCmd Cmd='Load' Bit='10' />
      <ExtractorCmd Cmd='Load' Bit='13' />
      <ExtractorCmd Cmd='Load' Bit='16' />
      <ExtractorCmd Cmd='Load' Bit='19' />
```

```
            <ExtractorCmd Cmd='Load' Bit='22' />
            <ExtractorCmd Cmd='Load' Bit='25' />
            <ExtractorCmd Cmd='Load' Bit='28' />
            <ExtractorCmd Cmd='Load' Bit='31' />
            <ExtractorCmd Cmd='Load' Bit='34' />
            <ExtractorCmd Cmd='Load' Bit='37' />
            <ExtractorCmd Cmd='Load' Bit='40' />
            <ExtractorCmd Cmd='Load' Bit='43' />
            <ExtractorCmd Cmd='Load' Bit='46' />
            <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
            <ExtractorCmd Cmd='JumpDone' />
         </ExtractorCmds>
      </ExtractorSequence>
```

<ExtractorSequences> Element

The `<ExtractorSequences>` element contains `<ExtractorSequence>` elements.

Children This element can have the following children: `<ExtractorSequence>` (see page 49).

Parents This element can have the following parents: `<ExtractorGrammar>` (see page 46).

Example

```
<ExtractorSequences>
   <ExtractorSequence>
      <ExtractorPatterns>
         <ExtractorPattern Value='b1XX' Width='3' Enabled='T' />
      </ExtractorPatterns>
      <ExtractorCmds>
         <ExtractorCmd Cmd='Load' Bit='2' />
         <ExtractorCmd Cmd='Load' Bit='5' />
         <ExtractorCmd Cmd='Load' Bit='8' />
         <ExtractorCmd Cmd='Load' Bit='11' />
         <ExtractorCmd Cmd='Load' Bit='14' />
         <ExtractorCmd Cmd='Load' Bit='17' />
         <ExtractorCmd Cmd='Load' Bit='20' />
         <ExtractorCmd Cmd='Load' Bit='23' />
         <ExtractorCmd Cmd='Load' Bit='26' />
         <ExtractorCmd Cmd='Load' Bit='29' />
         <ExtractorCmd Cmd='Load' Bit='32' />
         <ExtractorCmd Cmd='Load' Bit='35' />
         <ExtractorCmd Cmd='Load' Bit='38' />
         <ExtractorCmd Cmd='Load' Bit='41' />
         <ExtractorCmd Cmd='Load' Bit='44' />
         <ExtractorCmd Cmd='Load' Bit='47' />
         <ExtractorCmd Cmd='WriteLabelTime' Name='Idata' BitTime='0' />
         <ExtractorCmd Cmd='Load' Bit='1' />
         <ExtractorCmd Cmd='Load' Bit='4' />
         <ExtractorCmd Cmd='Load' Bit='7' />
         <ExtractorCmd Cmd='Load' Bit='10' />
         <ExtractorCmd Cmd='Load' Bit='13' />
         <ExtractorCmd Cmd='Load' Bit='16' />
         <ExtractorCmd Cmd='Load' Bit='19' />
         <ExtractorCmd Cmd='Load' Bit='22' />
         <ExtractorCmd Cmd='Load' Bit='25' />
         <ExtractorCmd Cmd='Load' Bit='28' />
         <ExtractorCmd Cmd='Load' Bit='31' />
         <ExtractorCmd Cmd='Load' Bit='34' />
```

```
            <ExtractorCmd Cmd='Load' Bit='37' />
            <ExtractorCmd Cmd='Load' Bit='40' />
            <ExtractorCmd Cmd='Load' Bit='43' />
            <ExtractorCmd Cmd='Load' Bit='46' />
            <ExtractorCmd Cmd='WriteLabel' Name='Qdata' />
            <ExtractorCmd Cmd='JumpDone' />
        </ExtractorCmds>
    </ExtractorSequence>
</ExtractorSequences>
```

Signal Extractor Commands

Command	Parameters	Description
Add2Regs	Number Second	Adds to one register (Number='0-15') a value from another register (Second='0-15'). <ExtractorCmd Cmd='Add2Regs' Number='2' Second='3' />
Add2RegsSignedLimit	Number Second Limit	Adds to one register (Number='0-15') a signed value from another register (Second='0-15'). The limit specifies the maximum and minimum values for the result (for example, with Limit='8', results must be from -80H to +7FH). <ExtractorCmd Cmd='Add2RegsSignedLimit' Number='2' Second='3' Limit='8' />
AddReg	Number Value	Adds to a register (Number='0-15') an unsigned 32-bit integer value. <ExtractorCmd Cmd='AddReg' Number='2' Value='hFFFF0' />
AddRegSignedLimit	Number Value Limit	Adds to a register (Number='0-15') a signed 32-bit integer value. The limit specifies the maximum and minimum values for the result (for example, with Limit='8', results must be from -80H to +7FH). <ExtractorCmd Cmd='AddRegSignedLimit' Number='2' Value='hFFFFFFC0' Limit='8' />
And2Regs	Number Second	Logically AND one register (Number='0-15') with a value from another register (Second='0-15'). <ExtractorCmd Cmd='And2Regs' Number='2' Second='3' />
AndReg	Number Value	Logically AND a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='AndReg' Number='2' Value='hFFFF0' />
DisablePattern	Number	Disables the specified pattern (Number='zero-based_number'). If a pattern is disabled, it is not available to be matched. <ExtractorCmd Cmd='DisablePattern' Number='0' />
Div2Regs	Number Second	Divides one register (Number='0-15') with a value from another register (Second='0-15'). <ExtractorCmd Cmd='Div2Regs' Number='2' Second='3' />
DivReg	Number Value	Divides a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='DivReg' Number='2' Value='hFFFF0' />
EnablePattern	Number	Enables the specified pattern (Number='zero-based_number'). If a pattern is enabled, it is available to be matched. The first pattern that matches the value is accepted. <ExtractorCmd Cmd='EnablePattern' Number='3' />
FindPulseWidth	None (but registers must be set up properly before command)	Calculates the smallest pulse width found on a specified input channel. See "Using the FindPulseWidth Command" on page 31. <ExtractorCmd Cmd='FindPulseWidth' />

Command	Parameters	Description
GoTo	Bit	Use this command to goto a specified bit (Bit='number'). This command is useful to position the extractor tool so that the next pattern match starts at the bit *after* the bit specified. Typically, you go to a specified bit and then execute a 'JumpDone' command. Note that a 'Load' bit command followed by a 'JumpDone' command has the same effect. <ExtractorCmd Cmd='GoTo' Bit='1024' />
GoToReg	Number	Go to a bit number defined by a register (Number='0-15'). <ExtractorCmd Cmd='GoToReg' Number='2' />
JumpBackward	Amount	Jump backward the specified number of commands (Amount='number' from 1 to N; 0 is not allowed). Use this command to change the execution by jumping backward in the command sequence. <ExtractorCmd Cmd='JumpBackward' Amount='2' />
JumpCase1Bit	Bit1	Use the value of the specified bit (Bit1='number') to determine the number of instructions to jump forward. If the value of the specified bit is zero, the next command will be executed. If the value of the specified bit is one, execution will skip one command and proceed on the next command. <ExtractorCmd Cmd='JumpCase1Bit' Bit1='0' />
JumpCase1BitReg	Reg1	This is equivalent to the JumpCase1Bit command except that the bit is loaded indirectly through the register. <ExtractorCmd Cmd='JumpCase1BitReg' Reg1='0' />
JumpCase2Bit	Bit1 Bit2	Use the value of the specified bits (Bit1='number', Bit2='number') to determine the number of instructions to jump forward. Note Bit1 is the most significant bit (MSB). If the value at Bit1 is 1 and Bit2 is 0, the Signal Extractor jumps forward 10 binary +1 (always jumps forward to next command) or a total of 3 commands. <ExtractorCmd Cmd='JumpCase2Bit' Bit1='0' Bit2='1' />
JumpCase2BitReg	Reg1 Reg2	This is equivalent to the JumpCase2Bit command except that the bits are loaded indirectly through the registers. <ExtractorCmd Cmd='JumpCase2BitReg' Reg1='0' Reg2='1' />
JumpCase3Bit	Bit1 Bit2 Bit3	Use the value of the specified bits (Bit1, Bit2, Bit3) to determine the number of instructions to jump forward. This command is similar to the 'JumpCase2Bit' command except that it uses 3 bits. <ExtractorCmd Cmd='JumpCase3Bit' Bit1='0' Bit2='1' Bit3='2' />
JumpCase3BitReg	Reg1 Reg2 Reg3	This is equivalent to the JumpCase3Bit command except that the bits are loaded indirectly through the registers. <ExtractorCmd Cmd='JumpCase3BitReg' Reg1='0' Reg2='1' Reg3='2' />
JumpCase4Bit	Bit1 Bit2 Bit3 Bit4	Use the value of the specified bits (Bit1, Bit2, Bit3, Bit4) to determine the number of instructions to jump forward. This command is similar to the 'JumpCase2Bit' command except that it uses 4 bits. <ExtractorCmd Cmd='JumpCase4Bit' Bit1='0' Bit2='1' Bit3='2' Bit4='3' />

Command	Parameters	Description
JumpCase4BitReg	Reg1 Reg2 Reg3 Reg4	This is equivalent to the JumpCase4Bit command except that the bits are loaded indirectly through the registers. <ExtractorCmd Cmd='JumpCase4BitReg' Reg1='0' Reg2='1' Reg3='2' Reg4='3' />
JumpCmp2Regs	Number Second	Compares one register (Number='0-15') with a value from another register (Second='0-15'). If register < value, execution jumps to the first command after. If register = value, execution jumps to the second command after. If register > value, execution jumps to the third command after. <ExtractorCmd Cmd='JumpCmp2Regs' Number='2' Second='3' />
JumpCmpReg	Number Value	Compares a register (Number='0-15') with an unsigned 32-bit integer value. If register < value, execution jumps to the first command after. If register = value, execution jumps to the second command after. If register > value, execution jumps to the third command after. <ExtractorCmd Cmd='JumpCmpReg' Number='2' Value='hFFFF0' />
JumpDone	None	Jump to the end of the command section and return to the pattern matching section of the Signal Extractor tool. <ExtractorCmd Cmd='JumpDone' />
JumpForward	Amount	Jump forward the specified number of commands (Amount='number' from 1 to N; 0 is not allowed). Use this command to change the execution by jumping forward in the command sequence. <ExtractorCmd Cmd='JumpForward' Amount='3' />
JumpTimeGreaterEqual	BitTimeStart BitTimeEnd TimePS	Lets Signal Extractor make decisions based upon the time between two bits. All bits of a particular sample have the same time stamp, so you must choose bit numbers that represent the sample that you want to test. If the time between the starting and ending bits is less than the time in picoseconds, execution jumps to the first command after. If the time between the starting and ending bits is greater than or equal to the time in picoseconds, execution jumps to the second command after. <ExtractorCmd Cmd='JumpTimeGreaterEqual' BitTimeStart='1' BitTimeEnd='40' TimePS='400' />
JumpTimeGreaterEqualRegs	Number Second TimePS	This is equivalent to the JumpTimeGreaterEqual command except that the bits whose timestamps are used are specified by register values. <ExtractorCmd Cmd='JumpTimeGreaterEqualRegs' Number='1' Second='2' TimePS='400' />
Load	Bit	Load the specified bit (Bit='number') into internal register 0. Bits already in the register are shifted by one bit to the left and this newest bit is ORed into the value. Note that bit 0 is considered the first left side bit in a pattern that matches. (Example: A 3 bit pattern is bit-0,bit-1,bit-2 for the pattern that matches and bit-3, bit-4, bit-5 for the next 3-bit pattern, etc.) <ExtractorCmd Cmd='Load' Bit='0' />

Command	Parameters	Description
LoadBitReg	Number	This is equivalent to the Load command except that the bit to load is specified by the value in the register (Number='0-15'). NOTE: The LoadRangeRegs command lets you load a range of bits as specified by two registers. The LoadRange command lets you load a range of bits as specified by two bit values. <ExtractorCmd Cmd='LoadBitReg' Number='3' />
LoadInit	None	Clears register 0. Register 0 is cleared when it is written to a label. It is also cleared at the beginning of the Signal Extractor. However, it is not cleared after a JumpDone command. The LoadInit command gives you the ability to use multiple extractor patterns to search for the bits to make a write statement. For example, these commands output 8 bits of Data from both patterns: ExtractorPattern Value='b1010xxxx' LoadRange BitStart='3' BitEnd='7' JumpDone ExtractorPattern Value='b1001xxxx' LoadRange BitStart='3' BitEnd='7' WriteLabelTime Name='Data' BitTime='0' Now, a second example using LoadInit to clear register 0 results in 4 bits of Data being output from just the second pattern: ExtractorPattern Value='b1010xxxx' LoadRange BitStart='3' BitEnd='7' JumpDone ExtractorPattern Value='b1001xxxx' LoadInit LoadRange BitStart='3' BitEnd='7' WriteLabelTime Name='Data' BitTime='0'
LoadOne	None	Loads a single bit of value one into internal register 0. <ExtractorCmd Cmd='LoadOne' />
LoadRange	BitStart BitEnd	Load the specified range of bits (BitStart='number', BitEnd='number') into internal register 0. Bits already in the register are shifted to the left and these newest bits are ORed into the value. Note that BitStart and BitEnd can be any positive value. The BitStart will be ORed in first regardless of whether BitStart is greater than or less than BitEnd. <ExtractorCmd Cmd='LoadRange' BitStart='0' BitEnd='7' />
LoadRangeRegs	Number Second	Load the specified range of bits where the starting bit in the range is in one register (Number='0-15') and the ending bit in the range is in another register (Second='0-15'). Bits already in register 0 are shifted to the left and these newest bits are ORed into the value. The starting bit will be ORed in first regardless of whether it is greater than or less than the ending bit. <ExtractorCmd Cmd='LoadRangeRegs' Number='2' Second='3' />
LoadReg (obsolete, replaced by MovReg)	Number Value	Loads a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='LoadReg' Number='2' Value='hFFFF0' />

Command	Parameters	Description
LoadZero	None	Loads a single bit of value zero into internal register 0. <ExtractorCmd Cmd='LoadZero' />
Mov2Regs	Number Second	Loads one register (Number='0-15') with a value from another register (Second='0-15'). <ExtractorCmd Cmd='Mov2Regs' Number='2' Second='3' />
MovReg	Number Value	Loads a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='MovReg' Number='2' Value='hFFFF0' />
Mult2Regs	Number Second	Multiplies one register (Number='0-15') with a value from another register (Second='0-15'). <ExtractorCmd Cmd='Mult2Regs' Number='2' Second='3' />
MultReg	Number Value	Multiplies a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='MultReg' Number='2' Value='hFFFF0' />
Or2Regs	Number Second	Logically OR one register (Number='0-15') with a value from another register (Second='0-15'). <ExtractorCmd Cmd='Or2Regs' Number='2' Second='3' />
OrReg	Number Value	Logically OR a register (Number='0-15') with an unsigned 32-bit integer value. <ExtractorCmd Cmd='OrReg' Number='2' Value='hFFFF0' />
ResetBitZero	None	Resets the zero location. This command saves you from having to remember bit locations. The bit zero location can be assigned to a new value and the relative bit locations are now based upon this new location. For example: GoTo Bit='97' ResetBitZero LoadRange BitStart='3' BitEnd='7'
Split	Amount Size Name	Use this command to split a pattern into specified number of patterns (Amount='2','4','8') of size (Size='number' of channels) and then place these values using appropriate time tags on the bus/signal (Name='string'). Use this command to Remultiplex A-to-D for fastest performance. Note: Additional qualifications may be needed for fastest performance. <ExtractorCmd Cmd='Split' Amount='4' Size='16' Name='A-to-D' />
Sub2Regs	Number Second	Subtracts from one register (Number='0-15') a value from another register (Second='0-15'). <ExtractorCmd Cmd='Sub2Regs' Number='2' Second='3' />
SubReg	Number Value	Subtracts from a register (Number='0-15') an unsigned 32-bit integer value. <ExtractorCmd Cmd='SubReg' Number='2' Value='hFFFF0' />

Command	Parameters	Description
`WriteLabel`	`Name`	Writes the internal register value to the specified bus/signal. Use this command to write out additional values to additional buses/signals. One `'WriteLabelTime'` command must be executed first to generate a new time stamp. This command writes to the 3 additional bus/signals of a common time stamp group. This command just fills in the values for additional bus/signals without generating additional samples. `<ExtractorCmd Cmd='WriteLabel' Name='Qdata' />`
`WriteLabelTime`	`Name` `BitTime`	Writes the internal register value to the specified bus/signal (`Name='string'`) and then copies the time associated with the specified bit (`BitTime='number'`) to the newly extracted value. After writing the internal register to a bus/signal the internal register is cleared. `<ExtractorCmd Cmd='WriteLabelTime' Name='Idata' BitTime='0' />` Be careful not to write multiple time values with the same time tag, and be careful not to write values such that time tags are out of sequence.
`WriteLabelTimeReg`	`Name` `Number`	This is equivalent to the `WriteLabelTime` command except that the bit whose timestamp is used is specified by a register value. `<ExtractorCmd Cmd='WriteLabelTimeReg' Name='Idata' Number='1' />`

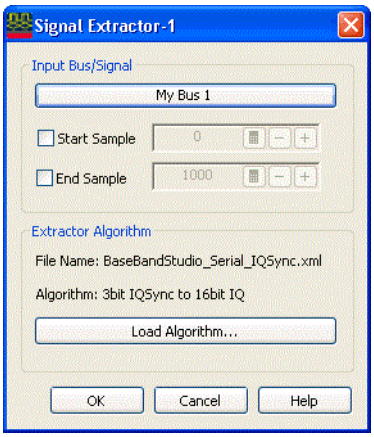
`WriteLabelTimeDelta`	`Name`
	`TimeNum`
	`TimeDen`
	`BitTimeStart`
	`BitTimeEnd`
		Writes the internal register value to the specified bus/signal (`Name='string'`) and then generates a time stamp by using a fraction (`TimeNum='decimal'`)/(`TimeDen='decimal'`) to create a time stamp that is the specified fraction between the `BitTimeStart='number'` and `BitTimeEnd='number'`. For example, to generate a time stamp that is half way between bit-29 and bit-30 you could use:
		`<ExtractorCmd Cmd='WriteLabelTimeDelta' Name='Idata' TimeNum='1' TimeDen='2' BitTimeStart='29' BitTimeEnd='30' />`

`WriteLabelTimeDeltaRegs`	`Name`
	`TimeNum`
	`TimeDen`
	`Number`
	`Second`
		This is equivalent to the `WriteLabelTimeDelta` command except that the bits whose timestamps are used are specified by register values.
		`<ExtractorCmd Cmd='WriteLabelTimeDeltaRegs' Name='Idata' TimeNum='1' TimeDen='2' Number='1' Second='2' />`

See Also · "<ExtractorCmd> Element" on page 43

8 Signal Extractor Tool Properties Dialog

The Signal Extractor tool properties dialog lets you set up the tool to extract signals or data from one input bus/signal and output the extracted data onto new buses/signals.



Input Bus/Signal	Selects the bus/signal that data will be extracted from.
Start Sample End Sample	Lets you reduce the number of samples being processed (and speed up processing).
Load Algorithm...	Lets you load the extractor algorithm that specifies the output bus/signal names and how data should be extracted. Example extractor algorithms have been included. You can find them under "Documents and Settings" for "All Users". For example, the default location is: C:\Documents and Settings\All Users\Shared Documents\Keysight Technologies\Logic Analyzer\Extractor Algorithms

See Also

- "Using the Signal Extractor Tool" on page 3
- Chapter 7, "Creating Extractor Algorithms," starting on page 21

9 Signal Extractor Tool Control, COM Automation

The *Keysight Logic Analyzer* application includes the COM Automation Server. This software lets you write programs that control the *Keysight Logic Analyzer* application from remote computers on the Local Area Network (LAN).

In a COM automation program, you can configure a tool by:

- Loading a configuration file (which configures the complete logic analyzer setup).
- Using the "Tool" (in the online help) object's "DoCommands" (in the online help) method with an XML-format string parameter (see Signal Extractor Tool Setup, XML Format (see page 63)).

You can get information about a tool's configuration using the Tool object's "QueryCommand" (in the online help) method. Queries supported by the Signal Extractor tool are listed below.

For more information about logic analyzer COM automation and tool objects in general, see "COM Automation" (in the online help).

XML-Based Queries Supported

The Signal Extractor tool supports the following XML-based queries (made with the "Tool" (in the online help) object's "QueryCommand" (in the online help) method).

Query	Description
GetAllSetup	Returns the current setup, using the full tag set, used for writing generic configuration files (see the XML format <Setup> element (see page 64)).

See Also

- "COM Automation" (in the online help)
- Chapter 10, “Signal Extractor Tool Setup, XML Format,” starting on page 63

10 Signal Extractor Tool Setup, XML Format

When you save logic analyzer configurations to XML format files, setup information for the Signal Extractor tool is included.

This XML format setup information is also used when writing COM automation programs to control the logic analyzer from a remote computer.

XML elements for the Signal Extractor tool have the following hierarchy:

```
<Setup> (see page 64)
```

See Also

- "XML Format" (in the online help)
- Chapter 9, “Signal Extractor Tool Control, COM Automation,” starting on page 61

<Setup> Element

The <Setup> element contains setup information for the Signal Extractor tool.

Attributes

Name	Description
EndingSampleNumber	'number'
ExtractorFile	'string'
InputBus	'string'
InputBusModule	'string'
StartingSampleNumber	'number'

Parents

This element can have the following parents: "<Tool>" (in the online help).

When used in COM automation, this element is returned by the "QueryCommand method" (in the online help)'s GetAllSetup query. You can also use this element string as an XMLCommand with the "DoCommands method" (in the online help) to configure the Serial To Parallel tool.

Example

```
<Setup InputBusModule='My 16950A-1' InputBus='My Bus 1'
      ExtractorFile='Z:\Documents and Settings\ab1234\My
      Documents\Keysight Technologies\Logic Analyzer\Config
      Files\Extractor_Test.xml'
      StartingSampleNumber='0' EndingSampleNumber='1000' />
```

Index