Tektronix®

Introduction to the Programming Interface Command Translator on Oscilloscopes

TECHNICAL BRIEF

In firmware versions v1.30 and higher, supported Tektronix oscilloscopes[1] come equipped with a programming, or programmatic, interface (PI) translator to help migrate legacy automation systems to new oscilloscope platforms. This unique, new capability allows new scopes to receive older, unsupported commands and translates them into modern commands. Translations for many commands for MSO/DPO5000, DPO7000, and MSO/DPO70000 oscilloscopes are already included in the default dictionary, but some automation applications will require custom translations. This document will explain the functions of the PI Translator, show how to add custom translations, and provide examples.

The PI Translator

The PI Translator is a simple system. It acts as a "dictionary" that contains a list of legacy commands and their modern counterparts. In addition to direct translation, entries can be configured to alias to multiple modern commands, add or preserve command arguments, and skip commands that are no longer needed.

The PI Translator runs passively in the scope application and does not require any installation in the remote client. It intercepts commands as they are being received by the scope, compares them to a set of supported translations, and sends the translated command on to the scope firmware.

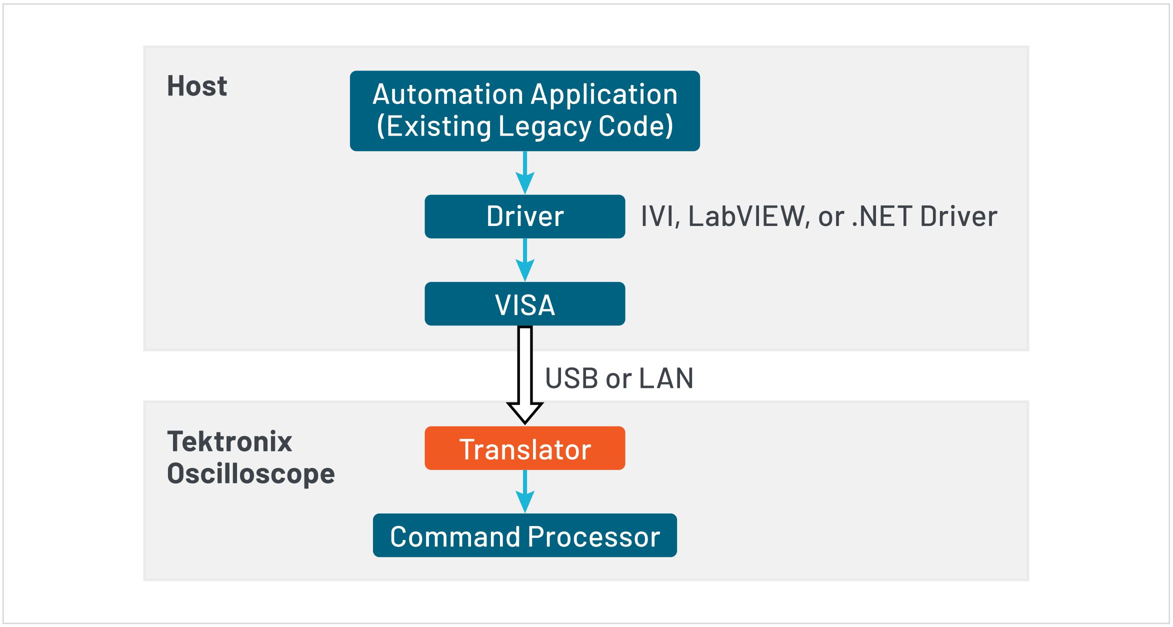

Host
Automation Application (Existing Legacy Code)
Driver
IVI, LabVIEW, or .NET Driver
VISA
USB or LAN
Tektronix Oscilloscope
Translator
Command Processor

Figure 1: The translator is built into the Tektronix oscilloscope and can convert "legacy" commands from the host to any standard commands.

By default, the PI Translator supports a large set of standard commands. It is intended to provide compatibility for common operations, like configuring horizontal, vertical, trigger, and acquisition settings, starting and stopping capture, and waveform transfer. This will cover many applications, but some automation sequences will require additional, custom commands to be added to the PI Translator.

The list of translations is contained in an XML file called the Compatibility File. This file must be edited to add new functionality to the PI Translator. XML files are simple, formatted text files. They can be edited in any text editor, such as Notepad or Microsoft Visual Studio. We prefer and recommend Notepad++ (https://notepad-plus-plus.org/), a free, open-source text editor that adds some functionality to make developing code easier.

Enabling the PI Translator

There are two ways to enable the PI Translator:

- Activate it in the oscilloscope's user interface by clicking/tapping the Utility menu at the top of the scope application, then selecting User Preferences->Other and flipping the "Programmatic Interface Backward Compatibility" button to On.
- Activate it with a PI command by sending `COMPatibility:ENABLE 1`.

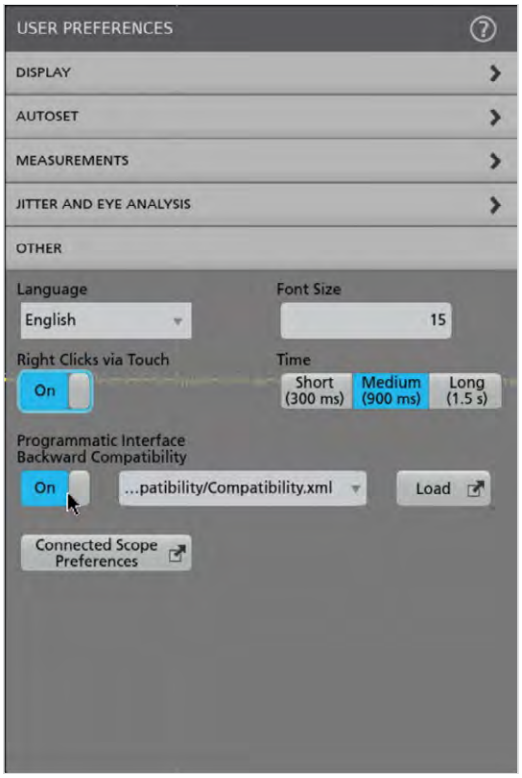
Once the PI Translator is enabled, it will translate commands according to a specified Compatibility File. The default file location when the oscilloscope is running the standard embedded Linux OS is:

C:/PICompatibility

The default location when the oscilloscope is running Windows is:

C:\Users\Public\Tektronix\TekScope\PICompatibility\Compatibility.xml

To modify the Compatibility File, we recommend that you copy Compatibility.xml and give it a new name. Then, to apply the changes to the PI Translator, choose your new Compatibility File using the file browser in the User Preferences menu by pressing the Load button, as seen in **Figure 2**. You can quickly switch between any loaded Compatibility Files and the default using the drop-down menu.



Language: English
Font Size: 15
Right Clicks via Touch: On
Time: Short (300 ms) | Medium (900 ms) | Long (1.5 s)
Programmatic Interface Backward Compatibility: On
...patibility/Compatibility.xml

Figure 2: Enabling the PI Translator in the scope UI

Anatomy of a PI Command

In order to effectively customize the Translator, it is important to understand how commands are constructed. All PI commands conform to a standard SCPI format. This means that they are composed of ASCII (text) strings followed by a termination character, 0x0A or '\n'.

Commands are typically constructed of a header, which tells the instrument what operation should be executed, and an argument, indicating which value or option should be applied to the operation.

Alternatively, some commands have a query form, indicated by a question mark (?) at the end of the header. This requests a response from the instrument, usually the current value of the setting. This response can then be read by the client program. Queries do not have arguments.

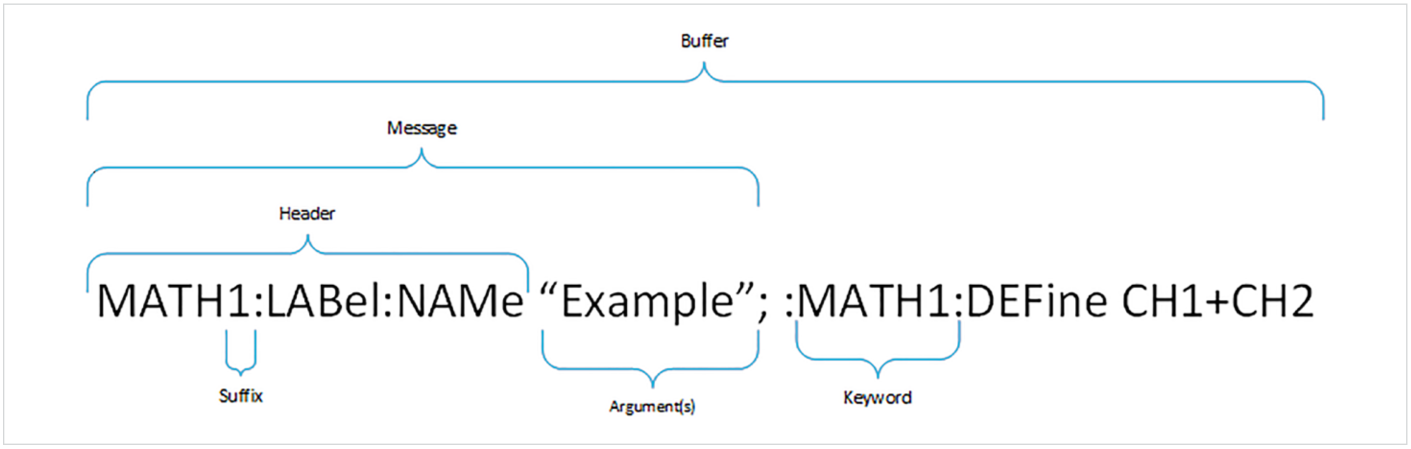


Figure 3: PI Command Structure

Buffer – The entire contents sent from the first character (leading colon or otherwise) to the newline character.

Message – Contents of a PI buffer from the first character (leading colon or otherwise) to either a semicolon (to denote a second message in a concatenated command) or a newline.

Header – Contents of a PI message that defines the action, specifically the string from the first character of a buffer to the first space, or the last character before a question mark.

Keyword – If a header is split by colons, a keyword would be a single element of the split list.

Suffix – Usually the trailing integer of a keyword (such as the `1` in `math:math1:define`) or a single character set apart by colons (as in `trigger:a:type`) Specifically, the suffix defines which instance of a particular type is used.

Argument – Arguments define what action or value a PI command should take. The format of this argument will vary from command to command and is defined in the Programmer Manual.

Command – When sending a message through the PI, a command indicates that something in the scope should change. Usually, but not always, this is a header followed by an argument.

Query – When sending a message through the PI, a query indicates that the scope must respond to a request. Always a header followed immediately with a question mark (?).

Translation Format

A PI Translator entry is composed of two parts: the keyword(s) from the original legacy command and the translation composed of supported commands. The legacy command should be split along each colon into keywords. For instance, `CURSor:WAVEform:POSITION2` would be split into `CURSor`, `WAVEform`, and `POSITION2` entries. Each keyword should have its own line in the compatibility file. Commands that share keywords can be grouped together, meaning that all `CURSor` commands can be listed under the same tree.

Each consecutive keyword in a command should be listed on a new line, with two spaces of indentation. Each line contains the **name** of the keyword and any number of optional **attributes**. Attributes are modifiers that indicate a keyword has special properties (for example, it could be a query or require an argument). Attributes are covered in detail in the next section, or you can skip to the Translation Examples section to see how to use them. The most important attribute is "leaf", which tells the Translator that this keyword is the end of a command. Every keyword with the "leaf" attribute should be followed by a translation containing a command found in the 2 Series MSO or the 4/5/6 Series MSO/LPD Programmer Manuals.

The translation should be listed, again, on a new line, with two spaces of indentation. This includes the **header**, which lists the supported command in full, and any number of optional attributes. These attributes are covered in detail in the next section.

Here is a simple, single-replacement entry:

```
<keyword name="HARDCOPY" leaf="1" command="1">
  <keyword name='FILENAME' leaf="1" command="1" query="1">
    <translation header=':SAVe:IMAGe:FULLPath'/>
  </keyword>
</keyword>
```

Considerations

To add a new translation, you should ask the following questions to help design the entry:

- What are all actions taken by the legacy command?
- What are the short and long forms of the legacy keywords?
- Is there a query form? A command form?
- What is the new header or headers?
- Does the argument of the legacy command matter?
- Does the suffix change what new header is used?
- Does the new header have suffixes that the previous header does not?
- Does the new header accept the same argument format as the legacy header?

Keyword Attributes

name
This is the only required portion of the entry. It should be entered in UPPERlower format, where the uppercase characters represent the valid short form of the keyword. (For example, SPECTral states that the scope will accept SPECT as a valid short form of the command 'spectral')

leaf
This attribute indicates that the entry is a valid stopping point for the Translator. For example, to translate "`MATH?:DEFine`", the "`DEFine`" node would have a "1" in the leaf attribute. If omitted, it is assumed that leaf equals zero and that more keywords will follow below. Note, a keyword can have the leaf attribute and still have following entries, but it must also have at least one translation entry.

command
This attribute defines if there is a valid command form for this PI header. Some PI headers are query only and this prevents the module from incorrectly handling the message. If omitted, it is assumed that command equals zero and no command handling will be performed on this header.

query
This attribute defines if there is a valid query form for this PI header. Some PI headers are command-only, and the scope cannot provide a response. If omitted, it is assumed that query equals zero and no query handling will be performed on this header.

argument
This attribute signals the module that there is a "sensitive argument" for this PI command. This means that the translation depends on both the header and the argument. This should be paired with a sensitiveArgument, addedArgument, or reuseArgument keyword attribute.

specialSuffix
This attribute signals to the module that the translation for this PI command will change based on the suffix. If set to 1, the Translator will note the suffix and use it to help define the translation. If omitted, it is assumed that specialSuffix equals zero and the keyword will be stripped of any suffix information while processing the header.

Translation Attributes

header
This is the only required part of the entry. This defines what header the legacy command will be replaced with. The new header should lead with a colon, and have '?' in any space where a suffix is used.

addedArgument
This attribute, usually used in conjunction with reuseSuffix and sendInQuery, or with sensitiveArgument, informs the module that the translation header already has an argument attached and no further argument handling is required for this header. If omitted, it is assumed that addedArgument equals zero and argument handling will proceed as normal.

sendInQuery
This attribute indicates if a translation header should be included in the query form of the message. This is used when multiple translation headers are needed to emulate a legacy header. Usually, the application does not expect multiple responses to one query, so this should be disabled for all but one header in a multi-header translation. If omitted, it is assumed that sendInQuery equals one, and the header will be sent in the PI query.

sensitiveArgument
This attribute defines what argument must be present for this translation to be valid. For example, “trigger:a:type” is a command that works in both legacy and supported Tektronix oscilloscopes[1], but the glitch trigger no longer exists. Instead, the pulse width trigger is used. Therefore, a ‘translation’ node exists for “trigger:?:type” with a sensitive argument of “GLItch”. Note, the same UPPERlower form is used, as arguments can have a short form.

If there should be a default header to use if all sensitive arguments failed (that is different than the initial header, such as “trigger:a:pulse:class”) this should be listed as the last translation option for a keyword and should exclude this attribute. If omitted, it is assumed there is no sensitive argument, and if paired with other sensitiveArgument translation nodes under the same keyword, it is assumed to be the default header.

reuseArgument
This attribute is used when two or more headers are required to emulate the legacy header and all new headers require the same argument. This flag will inform the module to preserve the argument when sending consecutive translation headers. This attribute should be disabled on the last header in a multi-header translation to make sure the argument is correctly cleared. The countOfArguments attribute is REQUIRED when using this attribute. If omitted, it is assumed that reuseArgument equals zero, and the argument information will be cleared after executing the translation header.

countOfArguments
This attribute indicates the number of previous arguments that are applied when reuseArgument is used. When enabled, it will add the reused argument to the current translation header. If omitted, it is assumed that no arguments are reused.

Note: attributes for handling suffixes (such as the “`a`” in `trigger:a:mode`) can be found in the Reference Manual.

Translation Examples

Note: there are additional examples available in the PI Translator Manual.

One-To-One Translation

This is the standard format from which all compatibility changes are built. It is used when a single message in supported Tektronix oscilloscopes[1] can be used in place of a legacy message. In this example, the command to define a math waveform is to be translated from a legacy DPO7000 command that has the syntax:

```
MATH<x>:DEFine <QString>
MATH<x>:DEFine?
```

To a PI message in supported Tektronix oscilloscopes[1] that explicitly specifies the math command group and has the syntax:

```
MATH:MATH<x>:DEFine <QString>
MATH:MATH<x>:DEFine?
```

The format for the QString argument is identical in both command sets. The PI translator entry to convert from the legacy command to the equivalent command in supported Tektronix oscilloscopes[1] is:

```
<keyword name='MATH?'>
   <keyword name='DEFine' leaf="1" query="1" command="1">
     <translation header=':math:math?:define'/>
   </keyword>
</keyword>
```

In this example, `MATH<x>:DEFine` will be replaced with `math:math<x>:define`. Note that the suffix is replaced with a '`?`'. This entry will work for both commands and queries because both attributes are enabled.

One-To-Many Translation

This format is used when two or more PI messages are needed to emulate a legacy command. In this example, the command to set the number of math averages is to be translated from a legacy DPO7000 command which has the syntax:

```
MATH<x>:NUMAVg <NR1>
MATH<x>:NUMAVg?
```

to two messages in supported Tektronix oscilloscopes[1] that set the number of averages and ensures that math averaging is enabled. On the legacy scope, setting the number of averages implicitly activates math averaging, but this operation must be explicitly enabled on modern platforms. These messages have the syntax:

```
MATH:MATH<x>:AVG:WEIGht <NR1>
MATH:MATH<x>:AVG:WEIGht?
MATH:MATH<x>:AVG:MODE 1
```

The PI translator entry to convert from the legacy command to the equivalent command in supported Tektronix oscilloscopes[1] is:

```
<keyword name='MATH?'>
    <keyword name='NUMAVg' leaf="1" query="1" command="1">
      <translation header=':math:math?:avg:weight' reuseSuffix="1"/>
      <translation header=':math:math?:avg:mode 1' addedArgument="1" sendInQuery="0"/>
    </keyword>
</keyword>
```

In this example, the 'leaf' keyword has two translation headers. The attributes in the first header indicate that the suffixes in the message should be preserved for the following message. The attributes in the second indicate that no argument should be expected (note that the translation header includes an argument) and that it should not be sent if used in a query.

Argument-Dependent Headers

In some cases, the appropriate header in a supported Tektronix oscilloscope[1] cannot be determined from the legacy command alone, but also requires information from the argument. This tool supports both long and short argument forms (subtract vs sub for example, provided the UPPERlower format is used in the argument) as well as two or more valid arguments. Note: If no 'default' header is provided, the system will make no change if the legacy message is a query.

In this example, the command to set the probe input mode (usually used for TriMode Probes) is to be translated from a legacy DPO7000 command which has the syntax:

```
CH<x>:PRObe:INPUTMode {DEFault|DIFFerential|COMmonmode|A|B}
CH<x>:PRObe:INPUTMode?
```

To a supported Tektronix oscilloscope[1] command that has the same header syntax but accepts different arguments and has a syntax of:

```
CH<x>:PRObe:INPUTMode {A|B|C|D}
CH<x>:PRObe:INPUTMode?
```

The PI translator entry to convert from the legacy command to the equivalent supported Tektronix oscilloscope[1] commands is:

```
<keyword name='CH?'>
   <keyword name='PRObe'>
      <keyword name='INPUTMode' leaf="1" command="1" query="1" argument="1">
         <translation header= ":CH?:PRObe:INPUTMode D" addedArgument='1' reuseSuffix='1' SensitiveArgument=
'DIFFerential'/>
        <translation header= ":CH?:PRObe:INPUTMode C" sendInQuery='0' addedArgument='1'
SensitiveArgument='COMmon'/>
      </keyword>
   </keyword>
</keyword>
```

In this example, both the legacy command and the translated command have four valid arguments. However, two of the arguments do not match exactly. One solution is to “translate the argument” using the sensitiveArgument attribute. If `DIFFerential` is received, the module will use D as an argument. If `COMmon` is received, it will use C. Note, if none of the four listed arguments are used, the system will make no change, as there is not a “default” header available. If there is a default behavior wanted, simply add an additional translation node after the last argument dependent header and the system will use the final option.

Reuse Argument

This is used when a global setting in legacy scopes needs to be emulated on a supported Tektronix oscilloscope[1]. This allows a one-to-many compatibility change to set an argument across several parameters. Note: At this time, the module does not have the ability to query how many instances of a parameter exist, so it may be necessary to observe this value before implementing.

In this example, the command to set the global trigger level is to be translated from a legacy DPO7000 command which has the syntax:

```
TRIGger:{A|B}:LEVel <NR3>
TRIGger:{A|B}:LEVel?
```

To a sequence of commands that set the trigger level for each channel and has the syntax:

```
TRIGger:{A|B}:LEVel:CH<x> <NR3>
TRIGger:{A|B}:LEVel:CH<x>?
```

The PI translator entry to convert from the legacy command to the equivalent supported Tektronix oscilloscope[1] command is:

```
<keyword name="TRIGger">
  <keyword name="?">
    <keyword name="LEVel" leaf="1" command="1" query="0">
      <translation header=":trigger:?:level:ch1" reuseSuffix="1" reuseArgument="1" countOfArguments="1"/>
      <translation header=":trigger:?:level:ch2" reuseSuffix="1" reuseArgument="1" countOfArguments="1"/>
      <translation header=":trigger:?:level:ch3" reuseSuffix="1" reuseArgument="1" countOfArguments="1"/>
      <translation header=":trigger:?:level:ch4"/>
    </keyword>
  </keyword>
</keyword>
```

The reuseSuffix and reuseArgument attributes tell the Translator to preserve both the suffix and the argument to be used by the next translation. In this example, both are used for channels 1 through 3, but for channel 4, the information no longer needs to be preserved. The countOfArguments attribute is used in case multiple arguments need to be preserved. In this example, there is only one argument per command.

Summary

The PI Translator already contains support for a variety of legacy commands. However, most applications will need some customization to reach full command coverage. The documentation and examples provided in this technical brief provide all the information needed for most situations. If you have a particularly advanced or difficult automation scheme, please refer to the PI Translator manual.

1. Supported Tektronix oscilloscopes:
   2 Series MSO
   4 Series MSO
   5 Series MSO
   5 Series B MSO
   6 Series MSO
   6 Series B MSO
   MSO58LP
   LPD64

Contact Information:

Australia	1 800 709 465
Austria*	00800 2255 4835
Balkans, Israel, South Africa and other ISE Countries	+41 52 675 3777
Belgium*	00800 2255 4835
Brazil	+55 (11) 3530-8901
Canada	1 800 833 9200
Central East Europe / Baltics	+41 52 675 3777
Central Europe / Greece	+41 52 675 3777
Denmark	+45 80 88 1401
Finland	+41 52 675 3777
France*	00800 2255 4835
Germany*	00800 2255 4835
Hong Kong	400 820 5835
India	000 800 650 1835
Indonesia	007 803 601 5249
Italy	00800 2255 4835
Japan	81 (3) 6714 3086
Luxembourg	+41 52 675 3777
Malaysia	1 800 22 55835
Mexico, Central/South America and Caribbean	52 (55) 88 69 35 25
Middle East, Asia, and North Africa	+41 52 675 3777
The Netherlands*	00800 2255 4835
New Zealand	0800 800 238
Norway	800 16098
People's Republic of China	400 820 5835
Philippines	1 800 1601 0077
Poland	+41 52 675 3777
Portugal	80 08 12370
Republic of Korea	+82 2 565 1455
Russia / CIS	+7 (495) 6647564
Singapore	800 6011 473
South Africa	+41 52 675 3777
Spain*	00800 2255 4835
Sweden*	00800 2255 4835
Switzerland*	00800 2255 4835
Taiwan	886 (2) 2656 6688
Thailand	1 800 011 931
United Kingdom / Ireland*	00800 2255 4835
USA	1 800 833 9200
Vietnam	12060128

*** European toll-free number. If not accessible, call:** +41 52 675 3777

Rev. 02.2022

Find more valuable resources at TEK.COM