Keithley Instruments, Inc.
23xx Fast Transient Response Power Supply Instrument Driver Readme

1. Overview

Instrument Driver Technology	LabVIEW Plug and Play (project-style)
Manufacturer	Keithley Instruments, Inc.
Supported Language(s)	LabVIEW
Supported Model(s)	2302, 2302-PJ, 2303, 2303-PJ, 2304A, 2306, 2306-PJ, 2306-VS, 2308
Model(s) Tested	2302, 2303, 2304A, 2306, 2306-PJ, 2306-VS, 2308
Interface(s)	GPIB
Firmware Revision(s) Tested	2302: B17, 2303: A09, 2304A: A05, 2306: B17, 2306-PJ: B17, 2306-VS: B17, 2308: A01
Certified	No
NI Supported	No
Source Code Available	Yes
Driver Revision	1.0.0.0
Original Release Date	04/15/2010
Current Revision Date	04/15/2010

2. Required Software

Some software components need to be installed before using this instrument driver. The minimum versions of these components are listed below, and can be downloaded from the Download Site.

VISA 3.0 or later

LabVIEW 8.0 or later

Refer to the *LabVIEW Help* for more information about software requirements. You access the *LabVIEW Help* by selecting *Help»Search the LabVIEW Help*.

3. Known Issues

To report issues or provide feedback about this instrument driver, please send an email to applications@keithley.com.

There are no known issues with this driver.

4. Revision History

The latest version of this instrument driver can be downloaded at the Keithley Instruments website.

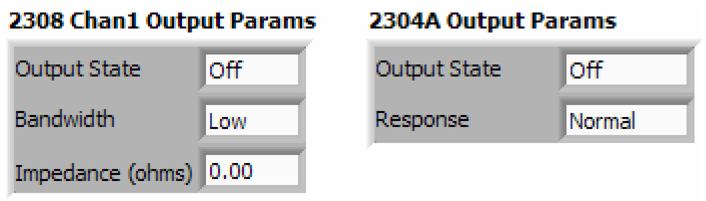
REV 1.0.0.0, 04/15/2010

Modified by: Keithley Instruments Inc., Cleveland, OH

Original release

5. Driver Usage

This driver makes heavy use of typedefs – almost every control / indicator on the VIs has a corresponding typedef, and these typedefs are typically instrument-specific. Typedefs are used because they are useful for parameter coercion (i.e. allowing the user to only enter values within a given parameter's range and minimum increment/granularity size), and in addition provide some level of type safety. For example, consider the case where a user tries to wire a control designed for the 2308 (call this the 'source' control) to a 2304A-specific VI input (call this the 'target' control), the connection will often be broken. This will typically occur when the source and target controls are clusters, but they are incompatible in the sense that they don't share the same elements. An example of this is trying to connect a 2308 'Output Params' control to a 2304A VI designed to set the output parameters. As the pictures of the corresponding control typedefs show, they are incompatible: the 2308 supports 'Output State', 'Bandwidth', and 'Impedance' as output parameters, while the 2304A supports 'Output State' and 'Response':



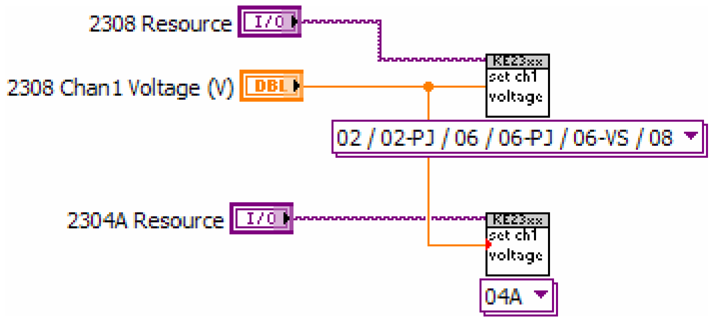
2308 Chan1 Output Params

Output State	Off
Bandwidth	Low
Impedance (ohms)	0.00

2304A Output Params

Output State	Off
Response	Normal

Note that even with controls that have the same base type, they are still individually typedef'ed if their valid *ranges* and/or the increment size aren't the same. For example, consider the voltage setting for the 2308 and the voltage setting for the 2304A, both with base type 'double'. The 2308 supports 0 to 15V, but the 2304A supports 0 to 20V. Shown below is an example where a voltage control typedef'ed for the 2308 is driving two instances of 'KE23xx – Set Ch1 Voltage.vi'. The top instance shows correct operation (polymorph selector set to 02 / 02-PJ / 06 / 06-PJ / 06-VS / 08), and the bottom instance shows incorrect operation (polymorph selector set to 04A).



In this case, LabVIEW will allow the connection, but it lets you know that a coercion has taken place via the red dot at the input. **In general, make sure any red coercion dots are not due to a typedef coercion – if they are, you are breaking the type safety provided with the typedefs.** So, to take advantage of the large number of instrument-specific typedefs, the suggested programming approach is to find the VI you need in the KE23xx palette, place it on the block diagram, then right-click on the inputs to create the controls via the pop-up menu (this makes it trivial to add controls since you don't have to search through a directory to find the typedef you are looking for). Finally, note that the range and increment coercion provided via the typedefs only works if the inputs to a given VI are driven directly from a front panel control. The implication here is that programmatically calculated values passed in to VIs *will not* be coerced by LabVIEW (e.g., suppose that the voltage passed into the above VIs was not directly from a front panel control, but was instead a calculated value calculated based on some other control). For values that are out-of-range, the instrument will report an error. However, the instrument *will not* report errors for values that violate the minimum increment size, but will instead silently coerce the value for you (e.g. trying to set the voltage of the 2304A to a 1mV resolution, where the actual minimum resolution is 5mV). Every VI has two optional controls to help deal with these situations, 'Check for SCPI Errors?' and 'Read Back Set Values?' (note that both default to True if left unconnected). If 'Check for SCPI Errors?' is True, an error query is sent to the instrument after the command is sent, and if 'Read Back Set Value(s)?' is True, the instrument will be queried for the actual *set* value, which can then be checked to see if the instrument performed any coercion (the set value is made available to the user via a VI output parameter). Note that both controls default to True for 'bullet-proofing'. Obviously, querying the instrument for errors and coercions comes with a speed penalty. So, it is recommended that user-created VIs have hooks to set these switch values to True or False (in the Example VIs, the switches are propagated up to the front panel). During initial program debug, it is best to leave these at their defaults (True). Once the user is satisfied that the VI is operating correctly, they can be set to False to speed-up program operation.

Finally, it is suggested that you only use the VIs that are in the palette set, at least when you are starting out (in particular, the Action-Status VIs are handy for getting something working quickly, since they handle many details ‘underneath the covers’). These are typically high-level VIs that set a number of parameters in the instrument via a single VI and a cluster control. Refer to the ‘Examples’ palette to see how the Action-Status VIs are used. Also note that almost every sub-palette has a ‘Test Code’ subpalette. If you are wondering whether or not a specific VI in a palette is working correctly, you can use the VIs in the ‘Test Code’ palettes without writing a piece of test code from scratch. Note that these are typically ‘set-then-get’ VIs, where the user-specified control parameters are first sent to the instrument, then these settings are ‘read back’ from the instrument and compared against what was sent. In this way, you can make sure that the parameters are being set properly in the instrument. Beware of issues when comparing floating point values – due to round-off errors in LabVIEW, it is possible that the ‘set’ value will not be exactly equal to the ‘get’ value. This occurs because when LabVIEW sends the value to the instrument via a string (the ‘set’ phase), it rounds the value to the specified instrument precision, which is always less than LabVIEW’s default double precision. Conversely, when it parses the response from the instrument (the ‘get’ phase), it converts this to the nearest double-precision number. Note that the resulting value after string parsing isn’t necessarily exactly equal to the original double precision value that LabVIEW used for the ‘set’ phase. In practice, the difference is very small (in fact, the controls will probably show the same value, but the ‘Set=Get?’ indicator on the test code VI front panel will be False). Furthermore, if you look at the actual *string* values being sent and received, they will be exactly the same – this implies that as far as the user is concerned, the value LabVIEW sent is exactly the same as what the instrument sent back.