Agilent U2300A Series USB Multifunction DAQ Devices VEE Application Program

Quick Reference Guide

Agilent Technologies

Safety Symbols

The following symbols indicate the precautions taken to maintain safe operation of the instrument.

	Direct current
	Warning

Regulatory Markings

	The CE mark shows that the product complies with all the relevant European Legal Directives (if accompanied by a year, it signifies when the design was proven).
	The CSA mark is a registered trademark of the Canadian Standards Association. A CSA mark with the indicators "C" and "US" means that the product is certified for both the U.S. and Canadian markets, to the applicable American and Canadian standards.
	The UL Mark is a registered trademark of Underwriters Laboratories Inc. UL listing mark with the indicators "C" and "US" indicates the product compliance with both Canadian and U.S. requirements.
	The C-tick mark is a registered trademark of the Spectrum Management Agency of Australia. This signifies compliance with the Australian EMC Framework regulations under the terms of the Radio Communications Act of 1992.

General Safety Information

WARNING

- **Do not use the device if it is damaged. Before you use the device, inspect the case. Look for cracks or missing plastic. Do not operate the device around explosive gas, vapor, or dust.**
- **Do not apply more than the rated voltage (as marked on the device) between terminals, or between terminal and external ground.**
- **Always use the device with the cables provided.**
- **Observe all markings on the device before connecting to the device.**
- **Turn off the device and application system power before connecting to the I/O terminals.**
- **When servicing the device, use only specified replacement parts.**
- **Do not operate the device with the removable cover removed or loosened.**
- **Do not connect any cables and terminal block prior to performing self-test process.**
- **Use only the power adapter supplied by the manufacturer to avoid any unexpected hazards.**

CAUTION

- Do not load the output terminals above the specified current limits. Applying excessive voltage or overloading the device will cause irreversible damage to the circuitry.
- Applying excessive voltage or overloading the input terminal will damage the device permanently.
- If the device is used in a manner not specified by the manufacturer, the protection provided by the device may be impaired.
- Always use dry cloth to clean the device. Do not use ethyl alcohol or any other volatile liquid to clean the device.
- Do not permit any blockage of the ventilation holes of the device.

Environmental Conditions

This instrument is designed for indoor use and in an area with low condensation. The table below shows the general environmental requirements for this instrument.

Environmental conditions	Requirements
Operating temperature	0 °C to 55 °C
Operating humidity	15% to 85% at 40 °C RH (non-condensing)
Storage temperature	–20 °C to 70 °C

CAUTION

The U2300A complies with the following safety and EMC requirements.

- IEC 61010-1:2001/EN 61010-1:2001 (2nd Edition)
- USA: UL61010-1: 2004
- Canada: CSA C22.2 No.61010-1:2004
- IEC/EN 61326-1 1998
- CISPR 11: 1990/EN55011:1991, Class A, Group 1
- CANADA: ICES-001: 1998
- Australia/New Zealand: AS/NZS 2064.1

Contents

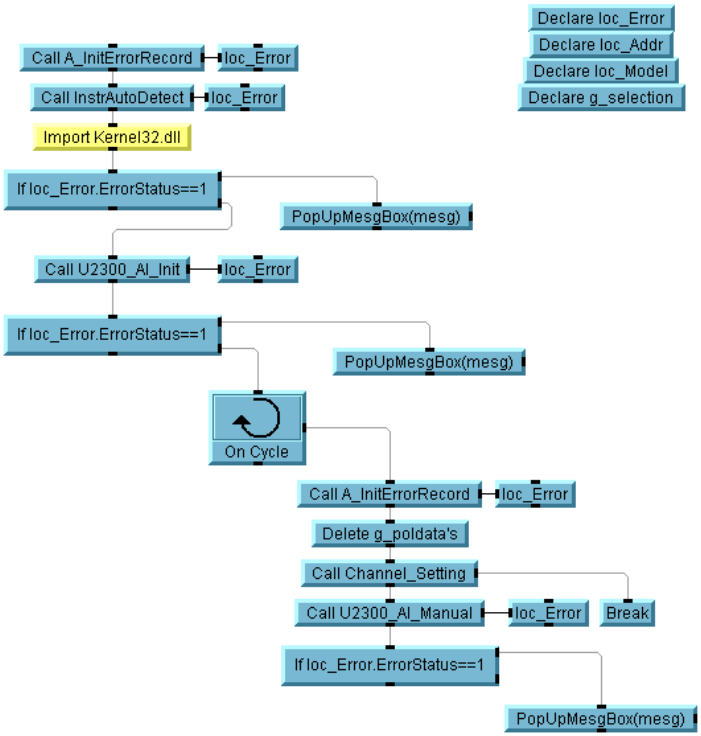
Introduction

This document will describe the functionality of each of the application. These applications consist of the following:

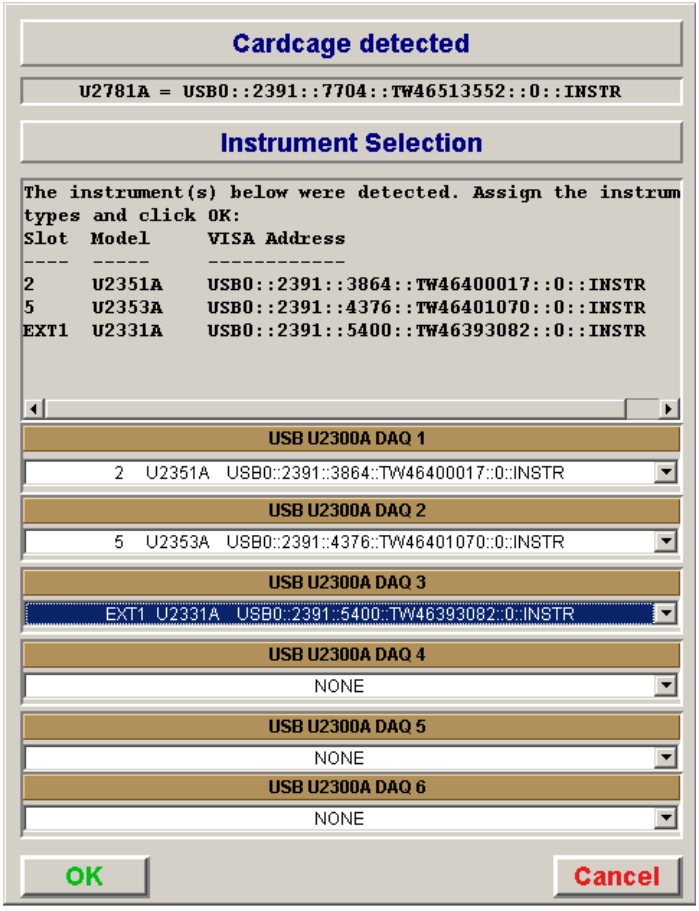
1. U2300_AI.vee — Analog input application
2. U2300_AO.vee — Analog output application
3. U2300_DIO.vee — Digital input/output application
4. U2300_CNT.vee — General purpose counter application
5. U2300_TempMonitor.vee — A temperature monitoring application that uses functions from "Simple U2300_AI.vee" and "Simple U2300_AO.vee".
6. U2300 Auto Prog Tool.vee — A simple test application that uses functions from "Simple U2300_AI.vee" and "Simple U2300_DIO.vee".
7. User-Defined Functions/Libraries

Analog Input (U2300_AI.vee)

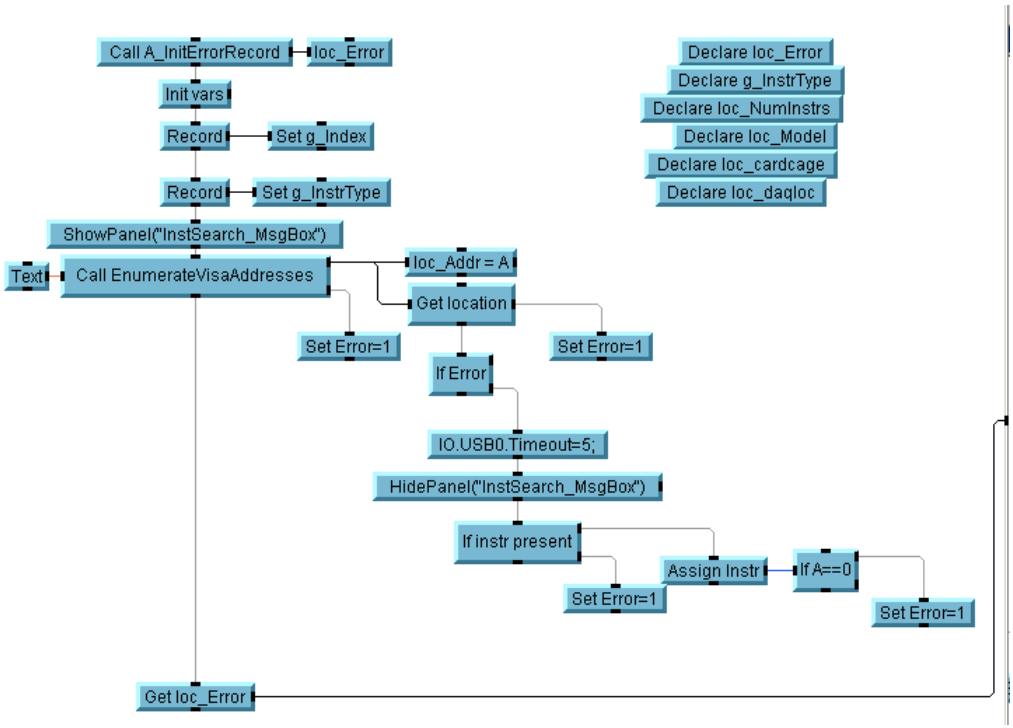
This is a standalone application that provides control over the analog input of the U2300A Series DAQ devices. The user can control up to a maximum of six units of DAQ within the cardcage or via any USB port. Users can monitor the input data on all the analog input channels. The program will show the enabled channels and it can show up to six graphs on the screen for the monitored data. Users can switch to view different channels and also display either a continuous waveform or a scatter diagram.



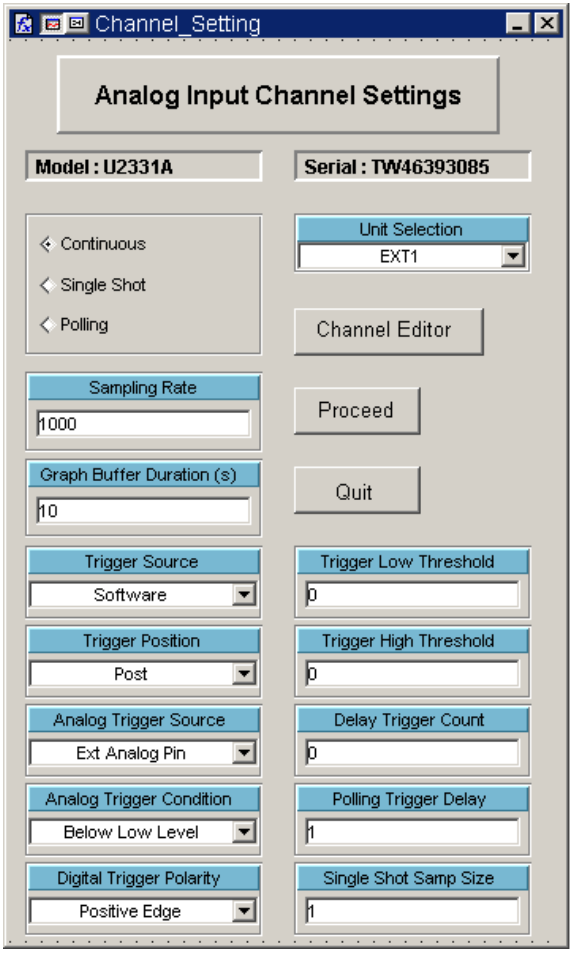
The main workspace is where all the main routine of the program is located. Once the program started, it will import the "kernel32.dll" file that will enable the program to read and write to initialize the files. It will then automatically detect any DAQ that is connected to the pc individually or in a cardcage. If it detects a cardcage, it will display the identification. It will also display all the DAQ that are either in the cardcage or on another USB port.



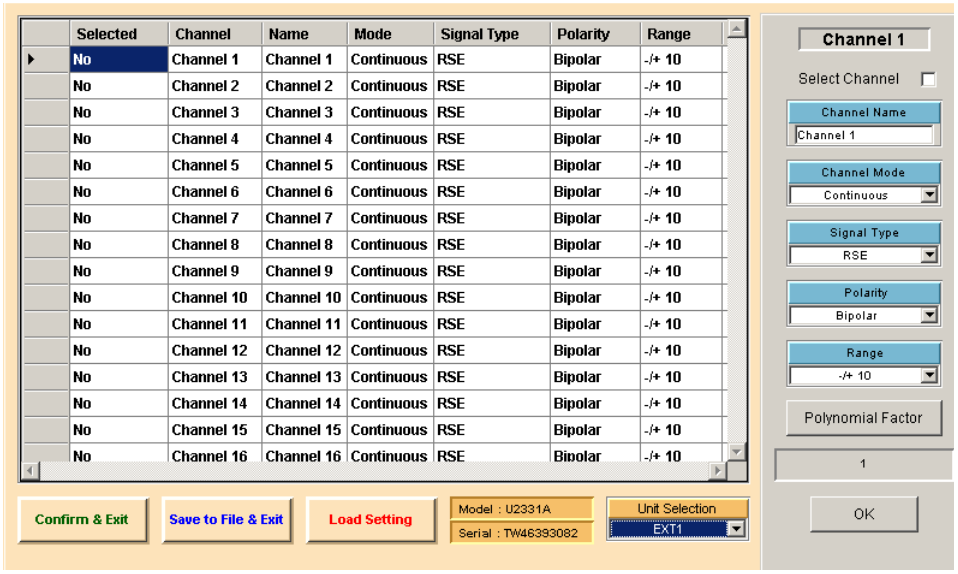
The detailed view of "InstrAutoDetect" is as shown below. It employs the "visa32.dll" to detect the DAQ that are connected to the pc.



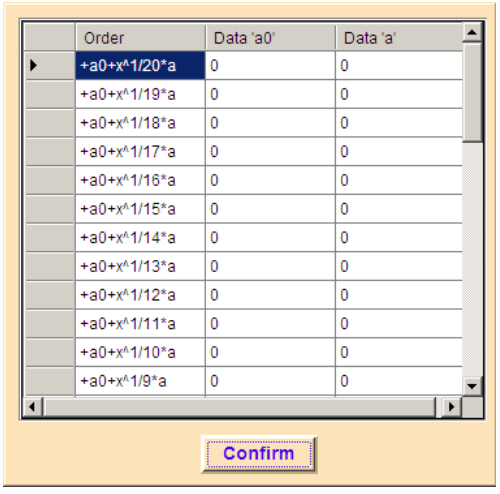
The program will then initialize all the program variables and establish communication with the instrument. It will obtain the serial and model number. It also checks for the maximum sampling rate and also how many analog input channels the attached unit has.



Once it has established the connection, the "Analog Input Channel Settings" screen will appear. This will enable the user to select the type of monitoring they want to perform, whether it is continuous mode, single shot mode or polling mode. The program will ask for the sampling rate they want to read the data. The user can set the triggering method and settings. Click the "Channel Editor" to configure which channel to run, otherwise the program cannot proceed.

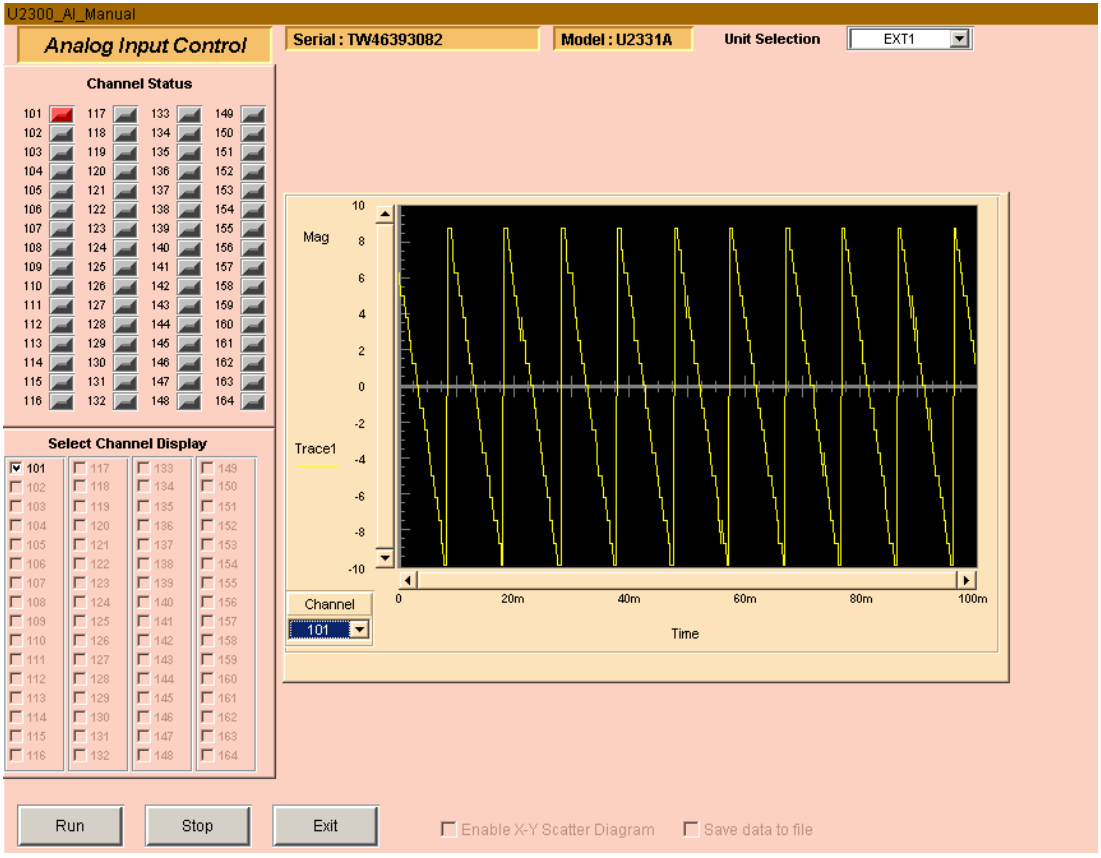


The channel editor will show the channels available depending on the attached model. Users can select each channel and set the mode, signal type, polarity, range and also the polynomial factor to be used for that channel. The settings can be saved to a Windows configuration settings file or "ini" files.



The monitoring screen will display the channel status, whether it is set to differential, RSE or NRSE. The user may select to view the graph for the enabled channels. A maximum of six graphs can be shown on the screen at any one time. The user may switch to look at the data at other channels by clicking the drop-down list next to each graph. By default, the program will display a waveform unless the user specify it to display an X-Y scatter diagram. Data can be saved to a "csv" (comma separated value) file

.



Channel status will show in bright red for the selected channels, dark red for the corresponding low-differential input and gray for disabled channels.

A list of all the userfunctions in the "U2300_AI.vee" application is as given;

No	Userfunction	Description
1	A_InitErrorRecord ()	Initializes the error handling variable record (loc_Error). Used for the functions that only sets the DAQ.
2	A_InitOutputRecord ()	Initializes the error handling variable of 6 records (loc_Output). Used for the functions that queries the DAQ.
3	A_InitSngOutputRecord()	Initializes the error handling variable of 1 record (loc_Output). Used for the functions that queries the DAQ.
4	Channel_Editor ()	Used as an internal function to allow users to select channels, set the monitoring mode, reference ground, etc. Uses the .NET datagrid function. Allows user to save and load the settings.
5	Channel_Popup (row)	Called by Channel_Editor()
6	Channel_Setting ()	Internal function to let user sets the sampling rate, monitoring mode, trigger settings, etc.
7	datagridView_MouseUp ()	Used by .NET datagrid
8	Disp_OneGraph () --> Disp_SixGraphs ()	Internal function to display the analog input data in graphical form. Able to display from one channel to six channels. User able to choose different channels to observe.
9	Disp_OneSChart () --> Disp_SixSCharts ()	Display the data in a strip chart format.
10	Disp_OneScatter () --> Disp_FourScatters ()	Display the data in a scatter diagram.
11	EnumerateVisaAddresses ()	Function that uses "visa32.dll" to detect all instruments that are connected to the pc.
12	Extract_ChanLoc (B)	Internal function to extract the channel data from an array.
13	FileSaving ()	Function that handles data file creation.
14	Get_BitBipData (Data, Bit, Rng)	Function that converts the bipolar data (byte) to decimal format for 12/16 bits.
15	Get_BitUniData (Data, Bit, Rng)	Function that converts the unipolar data (byte) to decimal format for 12/16 bits.
16	HandleVisaReturnCode ()	Function that handles error codes generated by "visa32.dll" while searching for instruments.
17	INI_Read (File, Section, Keyword)	Function that reads from "INI" file. Makes use of kernel32.dll.

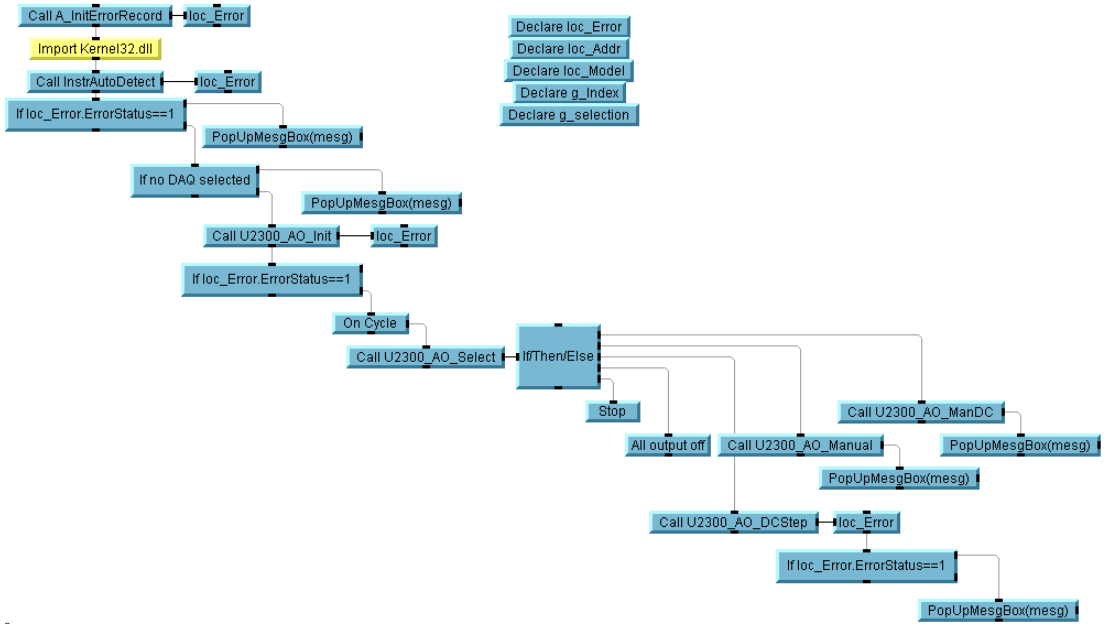
18	INI_Write (File, Section, Keyword, IniString)	Function that writes to "INI" file. Makes use of kernel32.dll
19	InstrAutoDetect ()	Function that automatically detects any instrument that is connected to the pc via GPIB, USB or LAN interface.
20	InstrInterfaceType (str)	Function that distinguish between a GPIB or USB interface.
21	InstSearch_MsgBox ()	Function that displays a message "Please wait... Searching for instrument"
22	LED_Display (BK1, BK2, BK3, BK4)	Internal function to display selected channels. Maximum channels = 64 Selected channel = Color "Warning Red" Unselected channel = Color "Dark Red" Disabled channel = Color "Dark Grey"
23	PopUpMesgBox (mesg)	Multipurpose display message box.
24	Select_Display (BK1, BK2, BK3, BK4)	Panel to allow user to display the channels they want to view while monitoring takes place. Maximum channels = 64. Separated into 4 inputs of 16 channels each.
25	String_IndexChar (string,char)	Finds the all the indices of the char in the string. Returns array of indices. If char doesn't exist in the string, this function returns -1. Limited to 100 indices.
26	U2300_AI_Init ()	This function must be called first to establish communication with the U2300 unit(s). It obtains the model and serial number and detects the numbers of channels and the maximum sampling rate.
27	U2300_AI_InitVar ()	This is called by U2300_AI_Init() function. It initializes all the variables used in this application.
28	U2300_AI_Manual ()	This function holds the main controls to the analog input application.
29	U2300_ConvertData (Select, Data, ChanPos)	Converts the data to the scaling that is input via the "Xth_Poly()" function.
30	U2300_Delay (X)	Common function for setting delay in seconds.
31	U2300_DigitizeUSB1~6 ()	Activate the U2300 "Digitize" function for each individual USB devices.
32	U2300_GetModel ()	Get the model of the attached U2300.
33	U2300_GetSerial ()	Get the serial number of the attached U2300.
34	U2300_GetWavCompUSB1~6 ()	Get the status of the availability of the waveform data for "digitize" mode.
35	U2300_GetWavDataUSB1~6 ()	Get the analog input data from the attached U2300 (Binary format).
36	U2300_Init ()	Perform a "Reset" and "Clear" the event registry and error queues of the U2300. Dynamically sets the interface address for the USB devices.

37	U2300_RunUSB1~6 ()	Activate the U2300 "Run" function.
38	U2300_ScanChan (Select, Chan)	Sets the channels for scanning on selected DAQ.
39	U2300_SetAnTrgCond (Select, Inp)	Sets the analog trigger condition on selected DAQ.
40	U2300_SetAnTrgHTrh (Select, Inp)	Sets the analog trigger high threshold value on selected DAQ.
41	U2300_SetAnTrgLTrh (Select, Inp)	Sets the analog trigger low threshold value on selected DAQ.
42	U2300_SetAnTrgSour (Select, Inp)	Sets the analog trigger source selection on selected DAQ.
43	U2300_SetChPolarity (Select, Pol, Chan)	Sets the channel polarity to bipolar or unipolar on selected DAQ.
44	U2300_SetChRange (Select, Range, Chan)	Sets the channel range on selected DAQ.
45	U2300_SetChType (Select, Stype, Chan)	Sets the reference ground for the channel on selected DAQ.
46	U2300_SetDigPol (Select, Inp)	Sets the polarity of the external digital trigger on selected DAQ.
47	U2300_SetSampRate (Select, MaxSamp, Chs, Inp)	Sets the sampling rate for the unit on selected DAQ.
48	U2300_SetTrgDCount (Select, Inp)	Sets the counter value for delay trigger mode on selected DAQ.
49	U2300_SetTrgSour (Select, Inp)	Sets the A/D trigger control source on selected DAQ.
50	U2300_SetTrgType (Select, Inp)	Sets the U2300 trigger mode on selected DAQ.
51	U2300_SetWavePoint (Select, Inp)	Sets the number of points for the waveform on selected DAQ.
52	U2300_StopUSB1~6 ()	Activate the U2300 "Stop" function.
53	U2300_TranslateData (Select, NoOfChans, ChanPos, data)	Converts the data from binary to decimal. Splits any interleave data from multiple channels.
54	Xth_Poly ()	Internal function that allows user to change the scaling factor.

Analog Output (U2300_AO.vee)

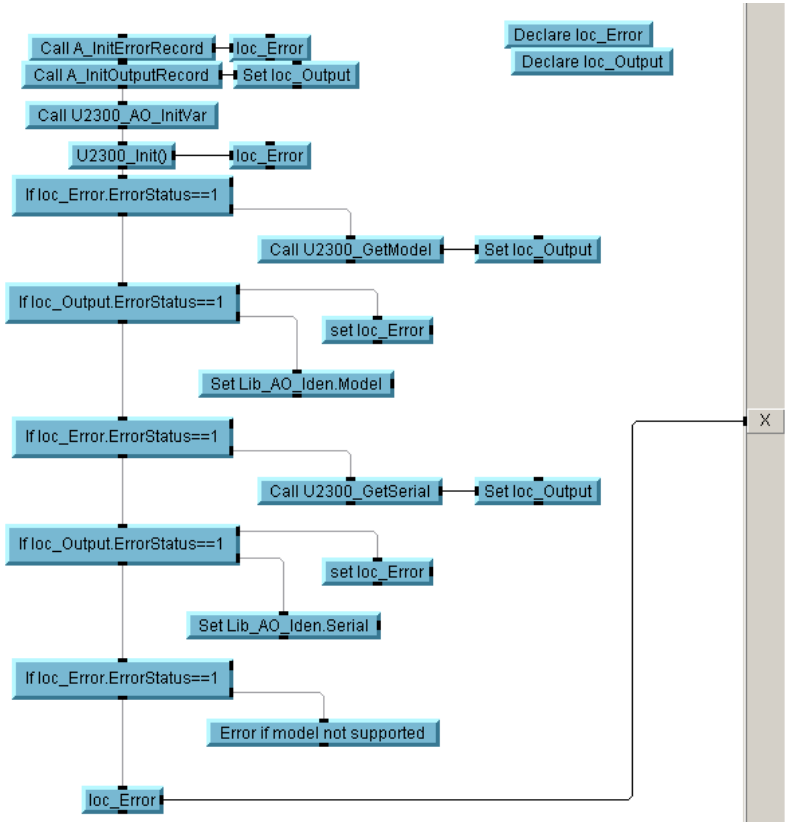
This is a standalone application that provides control over the analog output of the U2300A Series DAQ devices. The user can control up to a maximum of six units of DAQ within the cardcage or via any USB ports. Users can output dc voltages, standard waveforms and user-defined waveforms for analog output channel 201 and 202. The application will automatically detect the DAQ when it is connected to the pc via USB port.

The program will initialize all the program variables and establish communication with the instrument. It will obtain the serial and model number. It also checks if the attached U2300A Series DAQ devices support analog output. The following sequence is that it waits for the user to select DC voltage generation, standard waveform generation or DC step generation.

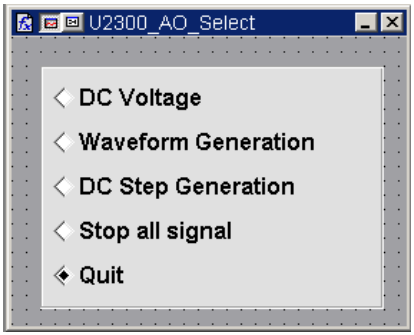


The main workspace is where all the main routine of the program is located. Once the program starts, it will import the "kernel32.dll" file that will enable the program to read and write to initialize the files. It then proceeds to automatically detect for any DAQ that is connected to the pc individually or in a cardcage.

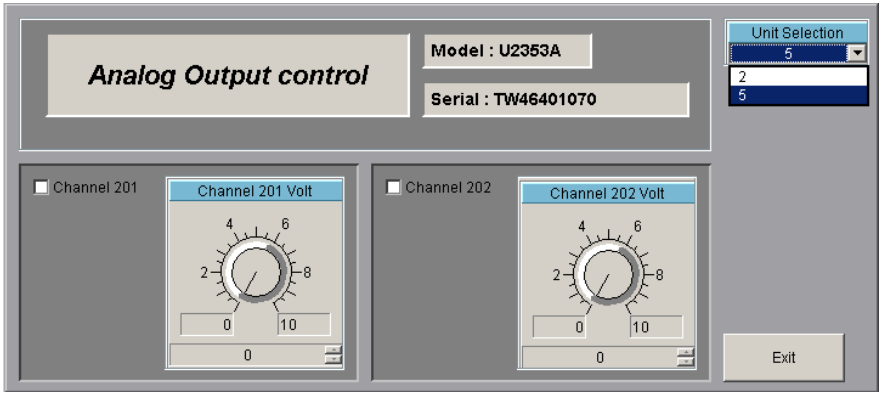
Once the DAQ selection is made, the program will initialize all the DAQ and get its model and serial number.



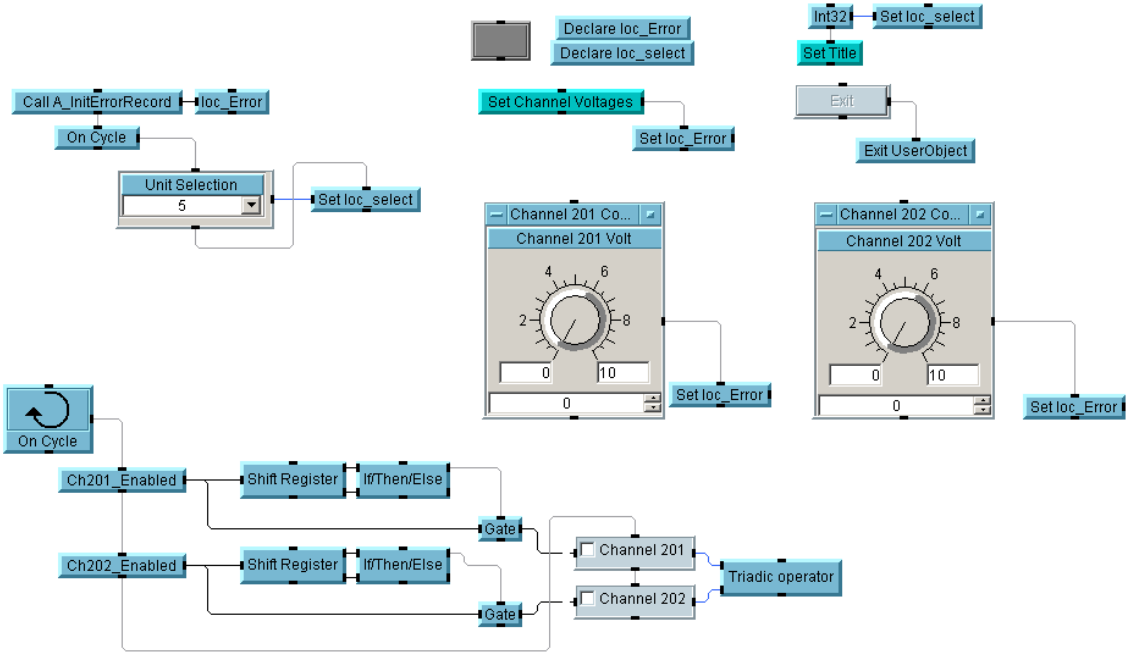
If no error is detected in the process, it will prompt the **U2300_AO_Select** dialog box for the user to select their choice of operation.



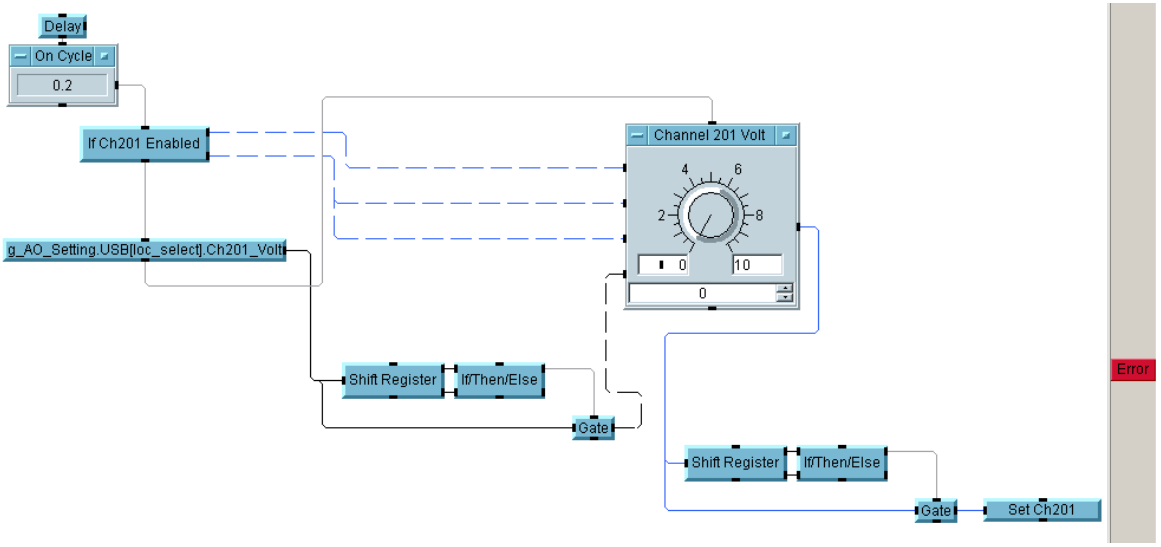
"DC Voltage" selection allows the user to simultaneously output voltage at channel 201 and 202 of any DAQ unit connected to the pc.



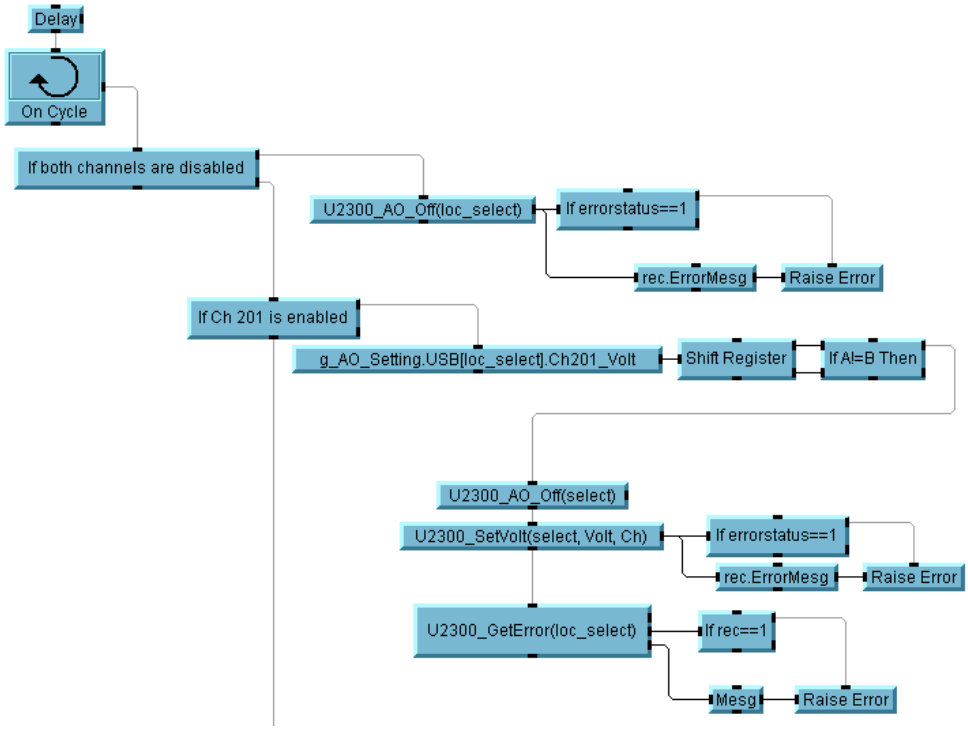
The "Analog DC Manual" control is made up from multiple user objects running in their own thread.



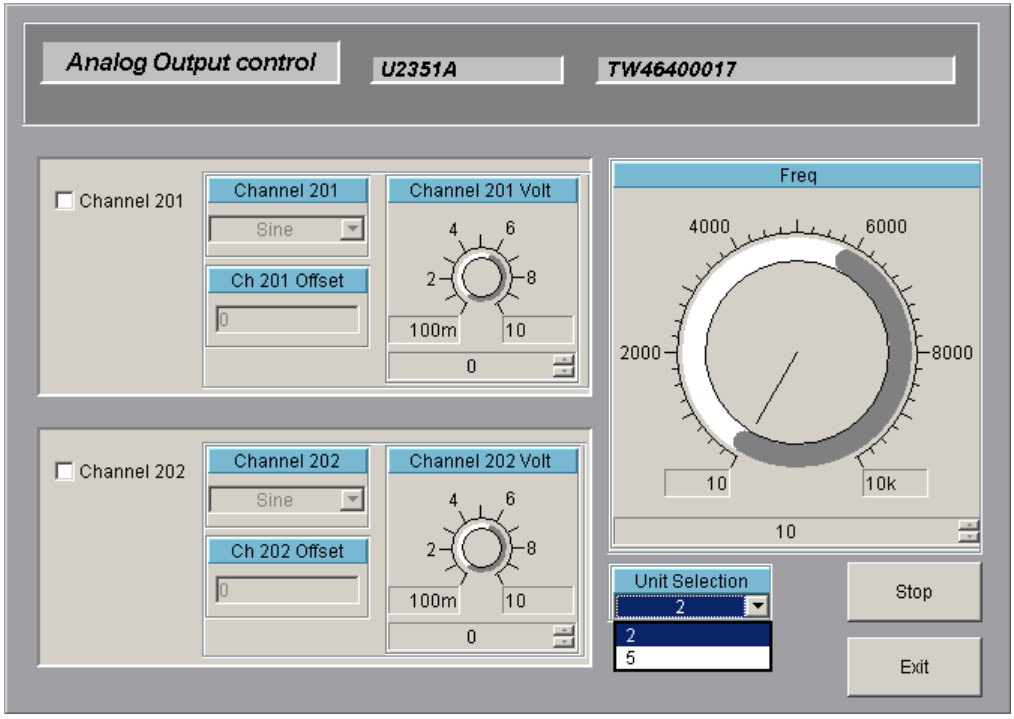
If we take a closer look at one of the channel control, we can observe that it will automatically update the voltage setting to a variable once the user turns the knob.



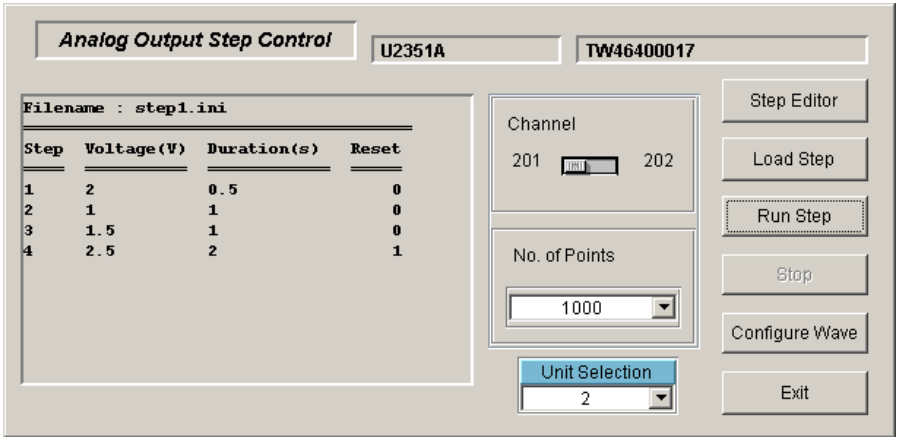
Once the variable is set, it will translate the data over to the DAQ in the "Set Channel Voltages" user-object.



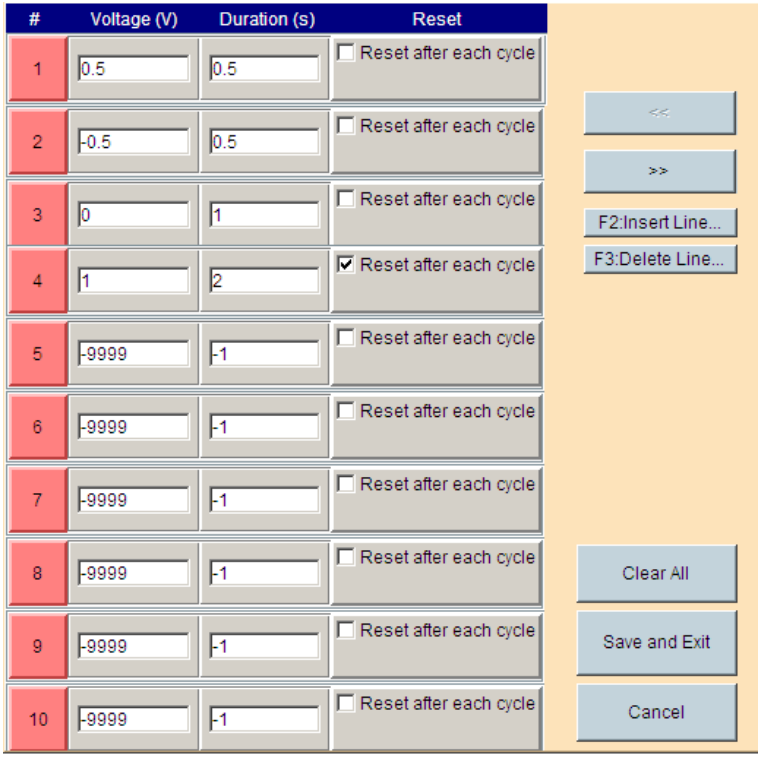
"Waveform Generation" has an array of waveforms such as sine, square, sawtooth, triangle and noise. User can simultaneously output different waveforms with varying amplitude and offset level for both channels. However, the frequency is fixed for both channels for each DAQ.



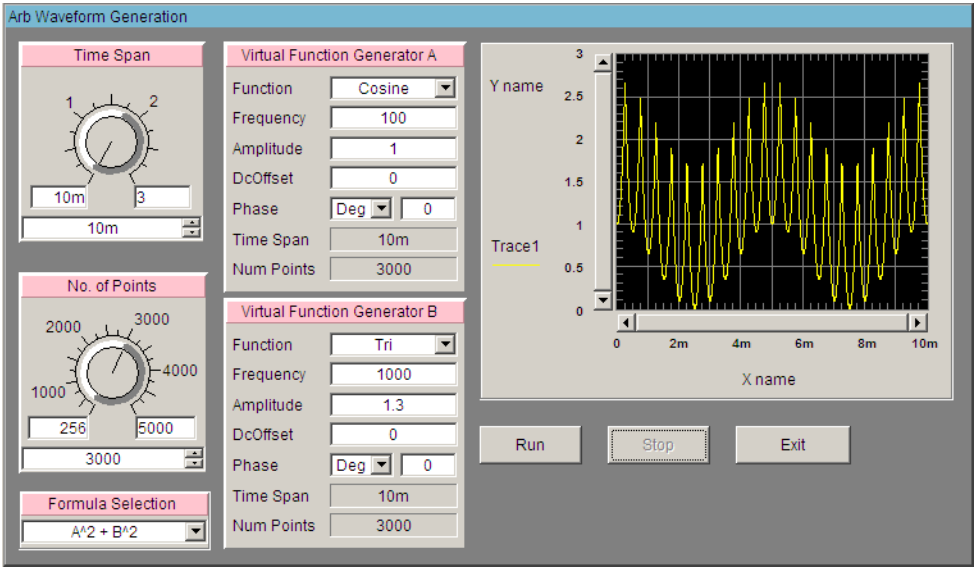
The "DC Step Generation" is a user-defined step voltage output. The user determines which channel to output the signal, the number of points and the step sequence. The application comes with a "Step Editor" that allows the user to configure the dc steps. There is a "Load Step" button to retrieve stored dc steps. Once the dc steps are loaded, clicking the "Run Step" button to initiate the pattern and "Stop" to halt the sequence.



The "Step Editor" can cater up to 100 steps. User can define the voltage level, the duration of each step and whether to set the voltage to 0 V once it finishes that step. A "–9999" on the voltage level represents a non-operational step, hence that step will be ignored. Users can save the configured steps into Windows configuration settings file or "ini" files.



The "Configure Wave" button lets the user create an arbitrary waveform. Users can combine waveforms from two virtual waveform generators through a series of predefined formulas. Users also have to determine the time span and the number of points before running it.



A list of all the userfunctions in the "U2300_AO.vee" application is as given;

No	Userfunction	Description
1	A_InitErrorRecord ()	Initializes the error handling variable record (Ioc_Error). Used for the functions that only sets the DAQ.
2	A_InitOutputRecord ()	Initializes the error handling variable of 6 records (Ioc_Output). Used for the functions that queries the DAQ.
3	A_InitSngOutputRecord()	Initializes the error handling variable of 1 record (Ioc_Output). Used for the functions that queries the DAQ.
4	AppendBackslash (path)	Appends a "\\" to the path name if it is not present in string.
5	Button_OnClick (Mode, Wildcard)	Internal function that allows the user to select a file (Reading or Writing mode will have to be given and also the wildcard extension). It will read from the current directory of this installed VEE file. It will output the selected file name and the path name.
6	Edit_Pattern ()	Internal function that allows user to visually set the step voltage for analog output. User can set the voltage, duration, reset to zero after each cycle and up to 100 steps. The settings can then be saved or retrieved from a configuration (*.ini) file.
7	EnumerateVisaAddresses ()	Function that uses "visa32.dll" to detect all instruments that are connected to the pc.
8	Editor_ArElemDelete (array, position)	Internal function that deletes a step in the analog output step editor.
9	Editor_ArElemInsert (array, position, newElem)	Internal function that inserts a step in the analog output step editor.
10	File_CheckExist (fname)	Function to check if the input file exists in the pc.
11	HandleVisaReturnCode ()	Function that handles error codes generated by "visa32.dll" while searching for instruments.
12	INI_Read (File, Section, Keyword)	Function that reads from "INI" file. Makes use of kernel32.dll.
13	INI_Write (File, Section, Keyword, IniString)	Function that writes to "INI" file. Makes use of kernel32.dll
14	InstrAutoDetect ()	Function that automatically detects any instrument that is connected to the pc via GPIB, USB or LAN interface.
15	InstrInterfaceType (str)	Function that distinguish between a GPIB or USB interface.

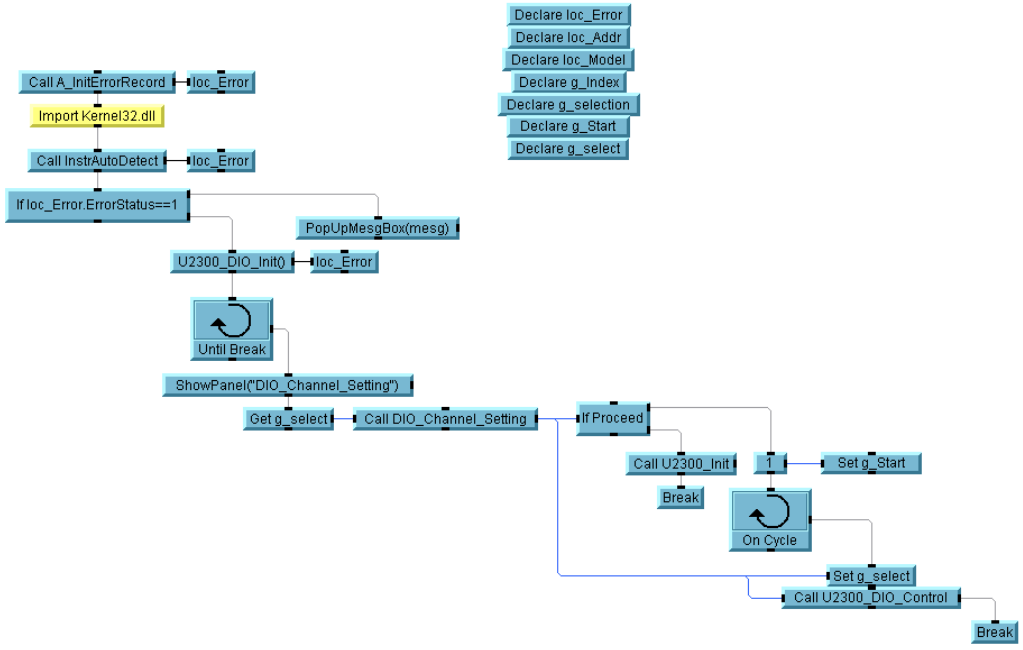
16	InstSearch_MsgBox ()	Function that displays a message "Please wait... Searching for instrument"
17	Int_ExtractStepRec (iniFile)	Internal function to extract the analog output steps from a given "ini" file. It formats the data to be displayed on the step editor.
18	Load_Pattern ()	Function that loads the analog output pattern from a selected "ini" file. Uses the Button_OnClick () function.
19	PopUpMesgBox (mesg)	Multipurpose display message box.
20	PopUpMesgBoxDec (mesg)	Display message box with a "YES" and "NO" button.
21	String_IndexChar (string,char)	Finds the all the indices of the char in the string. Returns array of indices. If char doesn't exist in the string, this function returns -1. Limited to 100 indices
22	U2300_AO_DCStep ()	This is the analog output step control screen. User can edit, load, run and stop the step patterns. User can choose from channel 201 or 202 to set the output from any DAQ. User can also set the number of points in which the program will calculate the sampling rate based on the total duration of the pattern and number of points. There is also an arbitrary waveform generator via the "Configure Wave" button. User can set the time span, number of points and combine the signals of two virtual function generator to output the waveform. There is also an "XY Trace" to display the combined waveform on the screen. The output channel can be selected from the previous screen.
23	U2300_AO_Init ()	This function must be called first to establish communication with the U2300 unit. It obtains the model and serial number and detects if the connected unit supports analog output functions.
24	U2300_AO_InitVar ()	This is called by U2300_AO_Init () function. It initializes all the variables used in this application.
25	U2300_AO_ManDC ()	This screen enables the user to set DC output voltage a single or dual channel (201 and 202)
26	U2300_AO_Manual()	This is an extended version of the U2300_AO_ManDC () function. It includes waveform type, offset setting and also the frequency for the fixed waveform (Sine, Square, Sawtooth, Triangle and Noise). User can set for single or dual channel.
27	U2300_AO_Off (Select)	Switches off the analog outputs on selected DAQ.
28	U2300_AO_Select ()	A panel displaying the selection for different applications.
29	U2300_GetError ()	Retrieve error messages from the U2300 unit.

30	U2300_GetModel ()	Get the model of the attached U2300.
31	U2300_GetOutput ()	Query the U2300 for the output voltage.
32	U2300_GetSerial ()	Get the serial number of the attached U2300.
33	U2300_Init ()	Perform a "Reset" and "Clear" the event registry and error queues of the U2300. Dynamically sets the interface address.
34	U2300_SetFixWave (Select, Type, Volt, Offset, Channel)	Sets the waveform type (Sine, Square, Sawtooth, Triangle, Noise), output voltage, voltage offset and the particular output channel (201 or 202) on the selected U2300 unit.
35	U2300_SetFreq (Select, Freq)	Sets the frequency for the waveform generation.
36	U2300_SetOutput (Select, Output)	Sets the analog output to "ON" or "OFF" depending on the input and selected DAQ.
37	U2300_SetUserData (Select, Ch, Header, Pattern, SRate)	Sets the "user's" analog output. Requires user to input the channel, header is like "#800000200, pattern is the binary format of output pattern and the sampling rate.
38	U2300_SetVolt (Select, Volt, Ch)	Sets the analog output voltage and on the selected channel and DAQ.
39	Universal_AOFormat (Pol, Bit, A)	Internal function that converts the given data (A) to binary format. User will have to provide the polarity ("BIP" [bipolar] or "UNI" [unipolar]) and also whether it is 12 or 16 bits.

Digital Input/Output (U2300_DIO.vee)

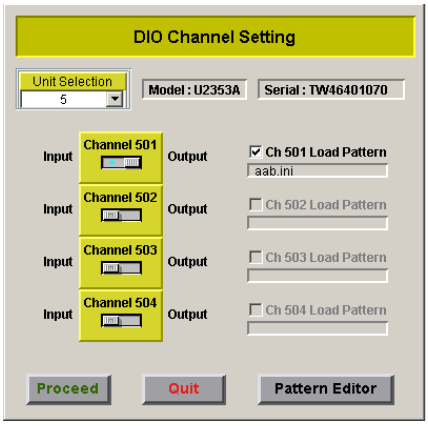
This is a standalone application that provides control over the digital port of the U2300A Series DAQ devices. The user can control up to a maximum of six units of DAQ within the cardcage or via any USB port. Users have control over four digital ports (Channel 501, 502, 503 and 504) and they can set it to be input or output. The application will automatically detects the DAQ when it is connected to the pc via USB port.

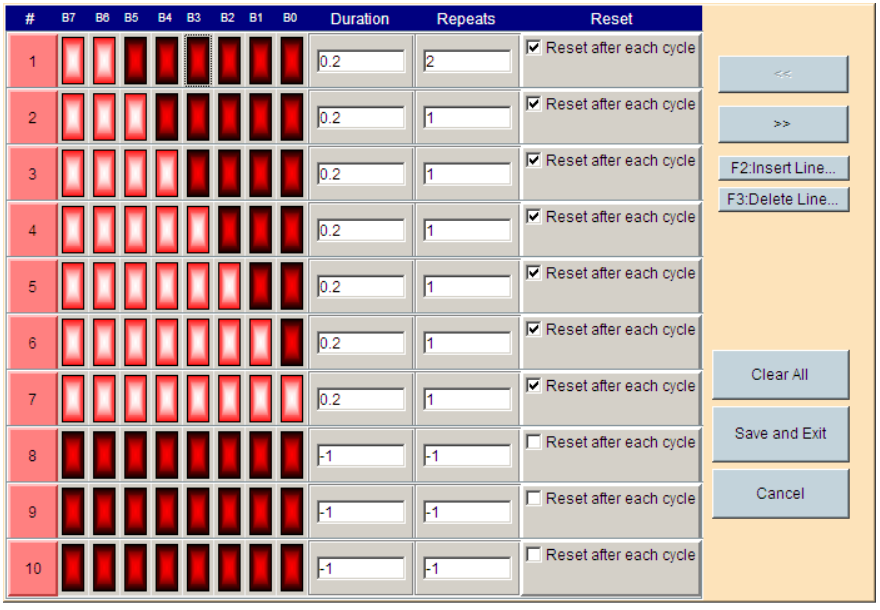
The program will initialize all the program variables and establish communication with the instrument. It will obtain the serial and model number. The main routine is located in the main workspace. Once the program starts, it will import the "kernel32.dll" file that will enable the program to read and write to initialize the files. It then proceeds to automatically detect for any DAQ that is connected to the pc individually or in a cardcage.



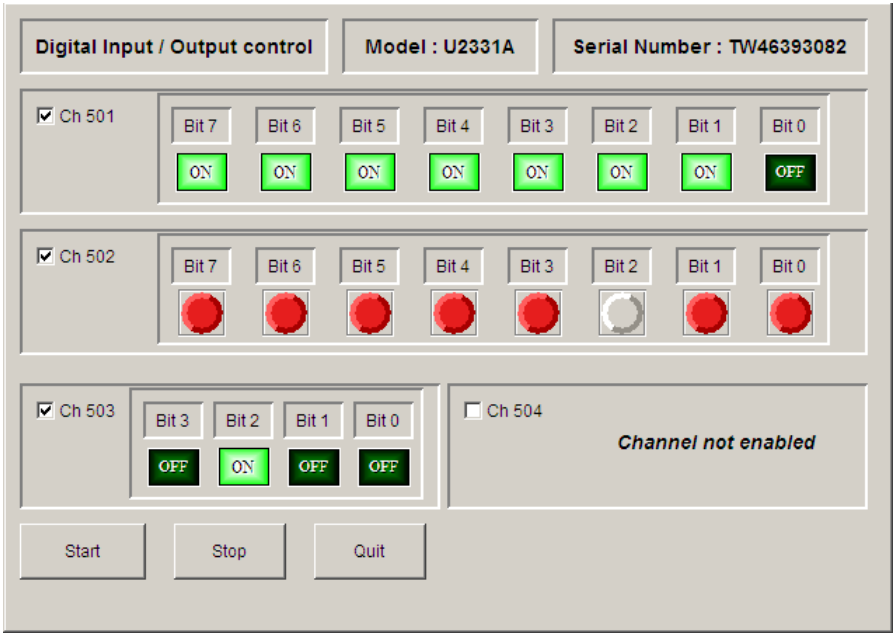
Once the DAQ selection is made, the program will initialize all the DAQ and get its model and serial number.

On the "DIO Channel Setting" screen, users can set the direction of the digital ports, they can retrieve pre-saved digital output steps and activate them via output ports. Users can click the "Edit Pattern" button to edit the digital output steps.



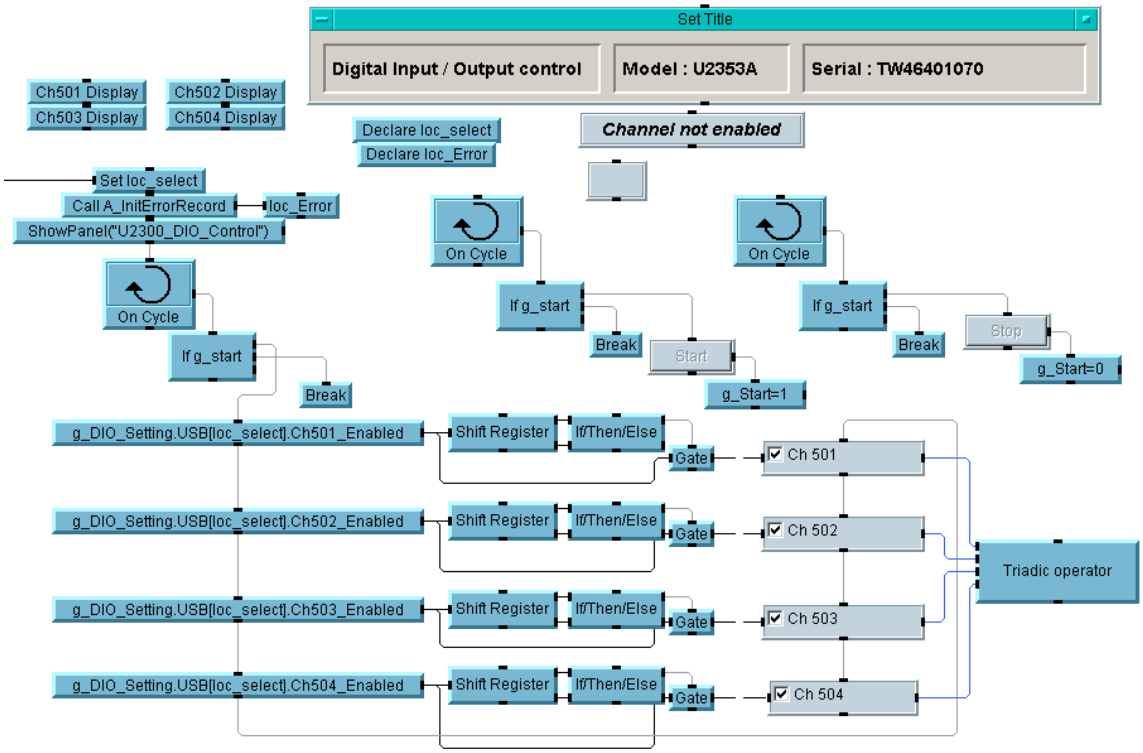


Within the pattern editor, users can define the output pattern bitwise, the duration, number of times to repeat and whether to reset after each step. It can cater up to 100 steps and a "–1" on the duration or repeats represents a non-operational step, hence it will be ignored. Users can save the configured steps into Windows configuration settings file or "ini" files.

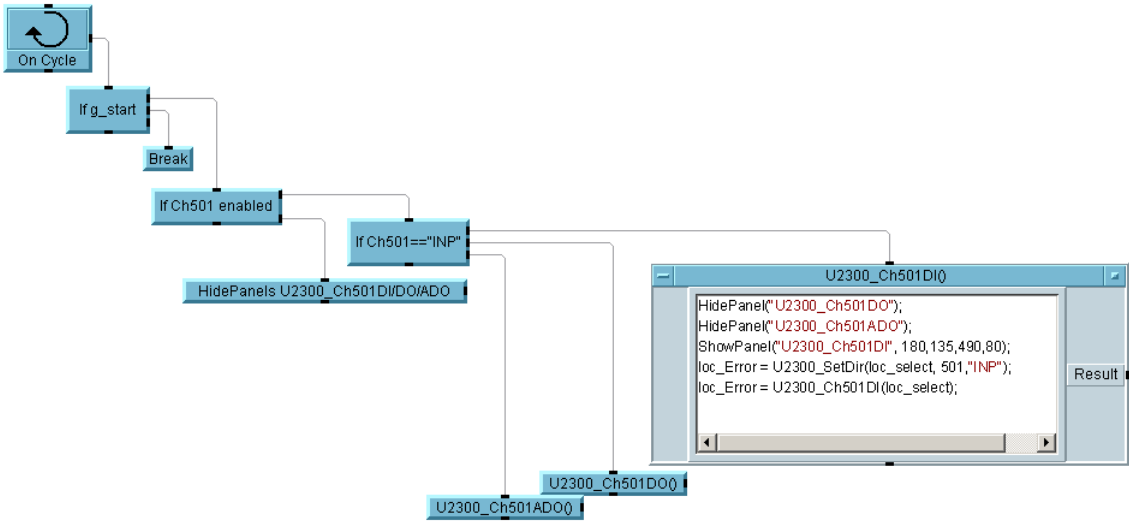


On the "Digital Input/Output control" screen, users have a choice to activate each channel. Based on the previously configured "DIO Channel Setting" screen, users can either control the output channels or view the input channels. The detailed view is as shown below. It has a few user-objects running in separate threads. The user's selection will determine what to display on the panel view.

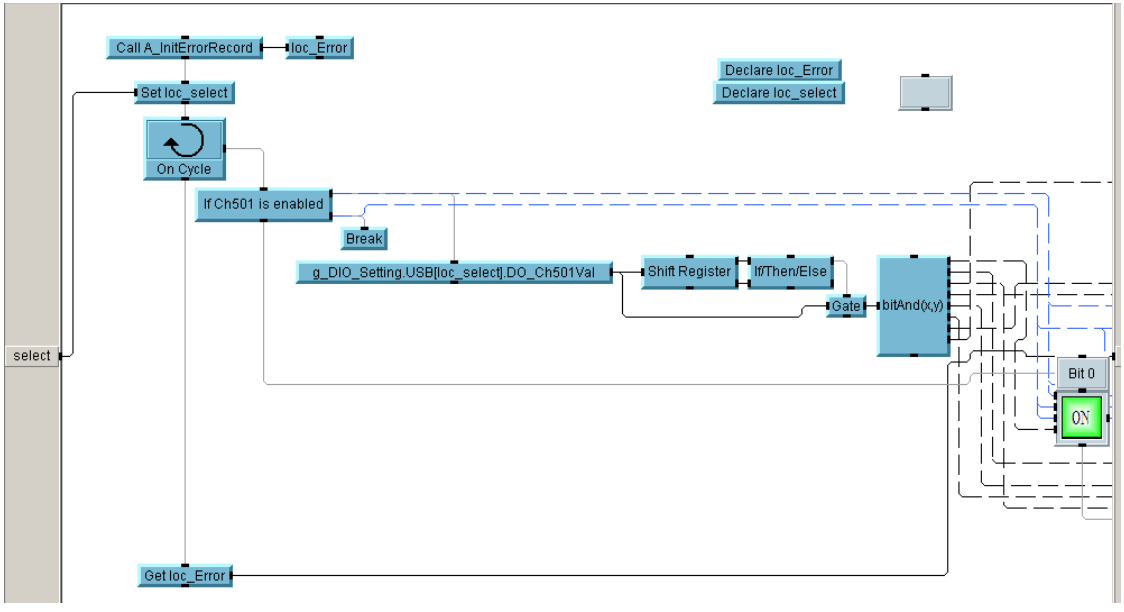
The controls on this user-object are to update the variables that are referred by other user-objects.

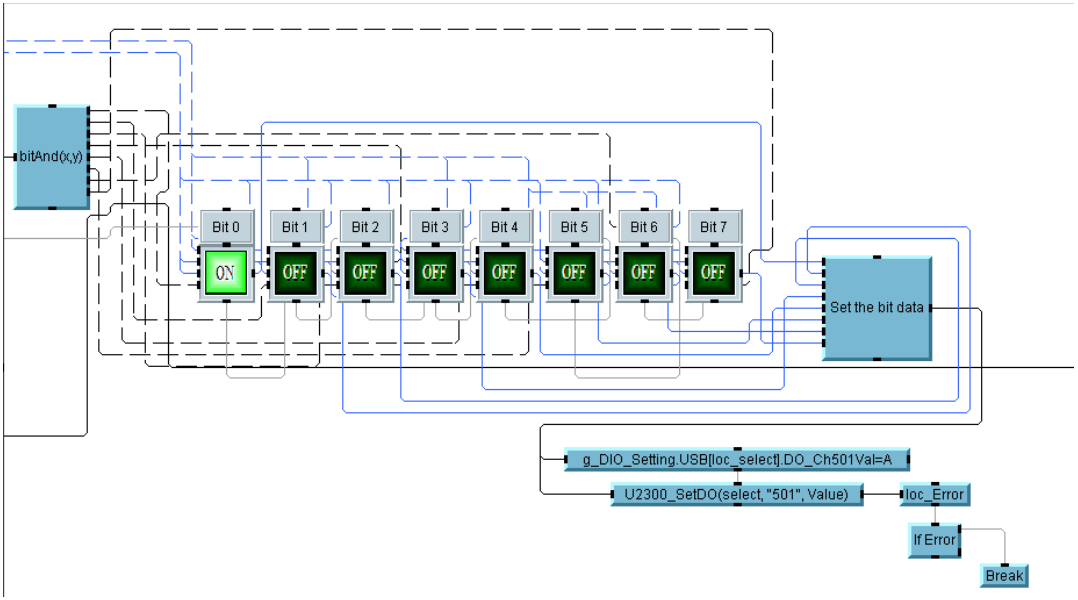


For example, the "Ch501 Display" user-object referred here is to determine what to display based on the user's selection.



The detailed view of "U2300_Ch501DO" shows the control of digital output for channel 501.




bitAnd(x,y)
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
ON
OFF
OFF
OFF
OFF
OFF
OFF
OFF
Set the bit data
g_DIO_Setting.USB[loc_select].DO_Ch501Val=A
U2300_SetDO(select, "501", Value)
loc_Error
If Error
Break

A list of all the userfunctions in the "U2300_DIO.vee" application is as given;

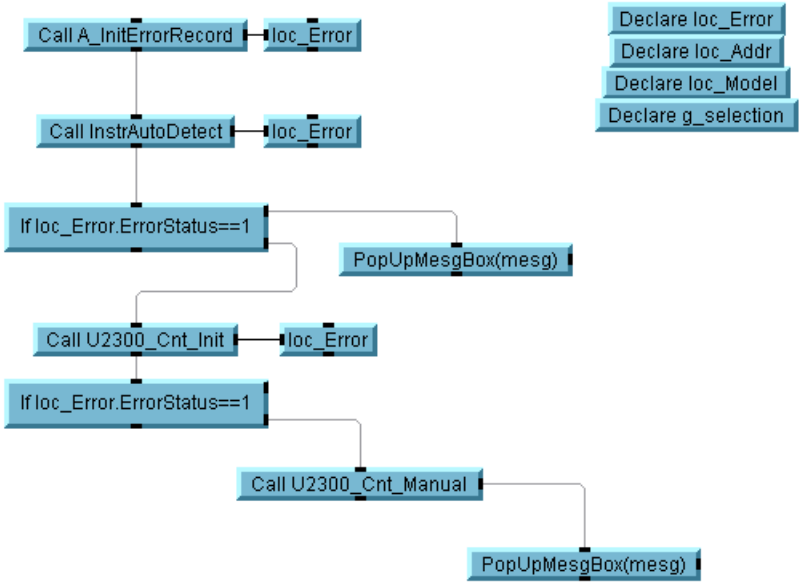
No	Userfunction	Description
1	A_InitErrorRecord ()	Initializes the error handling variable record (loc_Error). Used for the functions that only sets the DAQ.
2	A_InitOutputRecord ()	Initializes the error handling variable of 6 records (loc_Output). Used for the functions that queries the DAQ.
3	A_InitSngOutputRecord()	Initializes the error handling variable of 1 record (loc_Output). Used for the functions that queries the DAQ.
4	AppendBackslash (path)	Appends a "\\" to the path name if it is not present in string.
5	Button_OnClick (Mode, Wildcard	Internal function that allows the user to select a file (Reading or Writing mode will have to be given and also the wildcard extension). It will read from the current directory of this installed VEE file. It will output the selected file name and the path name.
6	DIO_Channel_Setting (Select)	This screen allows the user to set the digital channel to input or output direction. For output direction, it also allows you to select pre-configured digital output pattern files. The user can also set the digital output patterns via this screen by clicking on the "Pattern Editor" button. User can select any DAQ present.
7	Edit_Pattern ()	Internal function that allows user to visually set the state of each bit for each step. User can set the duration, number of repeats for each step, reset to zero after each cycle and up to 100 steps. The settings can then be saved or retrieved from a configuration (*.ini) file.
8	Editor_ArElemDelete (array, position)	Internal function that deletes a step in the analog output step editor.
9	Editor_ArElemInsert (array, position, newElem)	Internal function that inserts a step in the analog output step editor.
10	EnumerateVisaAddresses ()	Function that uses "visa32.dll" to detect all instruments that are connected to the pc.
11	File_CheckExist (fname)	Function to check if the input file exists in the pc.
12	HandleVisaReturnCode ()	Function that handles error codes generated by "visa32.dll" while searching for instruments.
13	INI_Read (File, Section, Keyword)	Function that reads from "INI" file. Makes use of kernel32.dll.
14	INI_Write (File, Section, Keyword, IniString)	Function that writes to "INI" file. Makes use of kernel32.dll

15	InstrAutoDetect ()	Function that automatically detects any instrument that is connected to the pc via GPIB, USB or LAN interface.
16	InstrInterfaceType (str)	Function that distinguish between a GPIB or USB interface.
17	InstSearch_MsgBox ()	Function that displays a message "Please wait... Searching for instrument"
18	Int_ExtractStepRec (iniFile)	Internal function to extract the digital output steps from a given "ini" file. It formats the data to be displayed on the step editor.
19	Load_Pattern ()	Function that loads the digital output pattern from a selected "ini" file. Uses the Button_OnClick () function.
20	PopUpMesgBox (mesg)	Multipurpose display message box.
21	PopUpMesgBoxDec (mesg)	Display message box with a "YES" and "NO" button.
22	String_IndexChar (string,char)	Finds the all the indices of the char in the string. Returns array of indices. If char doesn't exist in the string, this function returns -1. Limited to 100 indices
23	U2300_Ch501_ADO (Select) --> U2300_Ch504_ADO (Select)	This function is called when the digital output pattern generation is executed. It will display the state (ON or OFF) of each bit for that particular channel. Channel 501 and 502 are 8 bits whereas channel 503 and 504 are 4 bits.
24	U2300_Ch501_DI (Select) --> U2300_Ch504_DI (Select)	This function is called when the channel is set to input direction. It will show you the state of each bit of that particular channel.
25	U2300_Ch501_DO (Select) --> U2300_Ch504_DO (Select)	This function is called when the channel is set to output direction. You can manually click on each bit to enable or disable the bit of that channel.
26	U2300_DIO_Control (Select)	This function is the control screen for the user to view all the channels on selected DAQ.
27	U2300_DIO_Init ()	This function must be called first to establish communication with the U2300 unit. It obtains the model and serial number.
28	U2300_DIO_InitVar ()	This function initializes all the variables used in this program.
29	U2300_GetDI (Select, Ch)	Get the state of all the bits of the given channel (Ch) and selected DAQ.
30	U2300_GetModel ()	Get the model of the attached U2300.
31	U2300_GetSerial ()	Get the serial number of the attached U2300.
32	U2300_Init ()	Perform a "Reset" and "Clear" the event registry and error queues of the U2300. Dynamically sets the interface address.

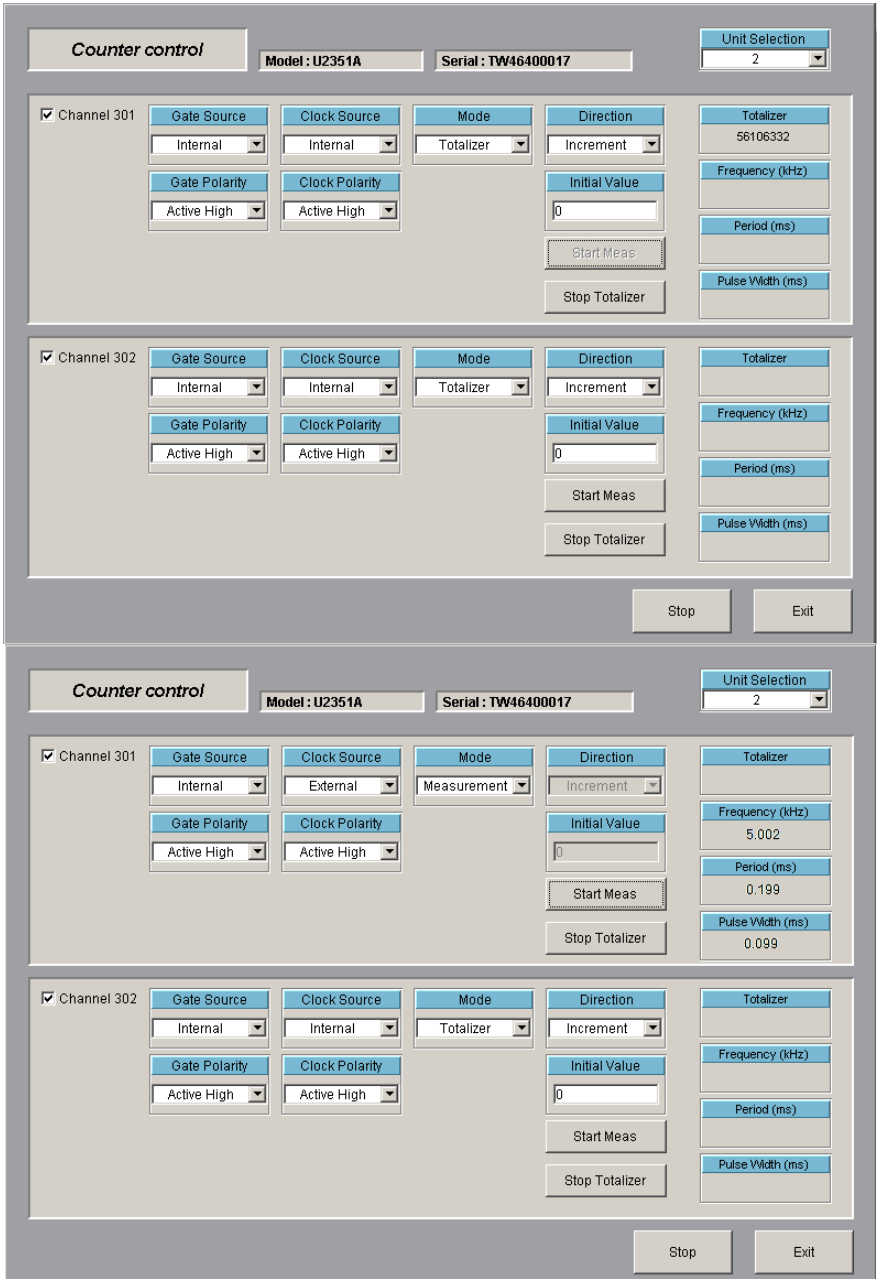
33	U2300_SetDir (Select, Ch, Dir)	Configures the given channel for either input ("INP") or output ("OUTP") direction on the selected DAQ.
34	U2300_SetDO (Select, Ch, Value)	Sets the given channel (Ch) to output the state based on the integer value given on the selected DAQ.

Counter (U2300_CNT.vee)

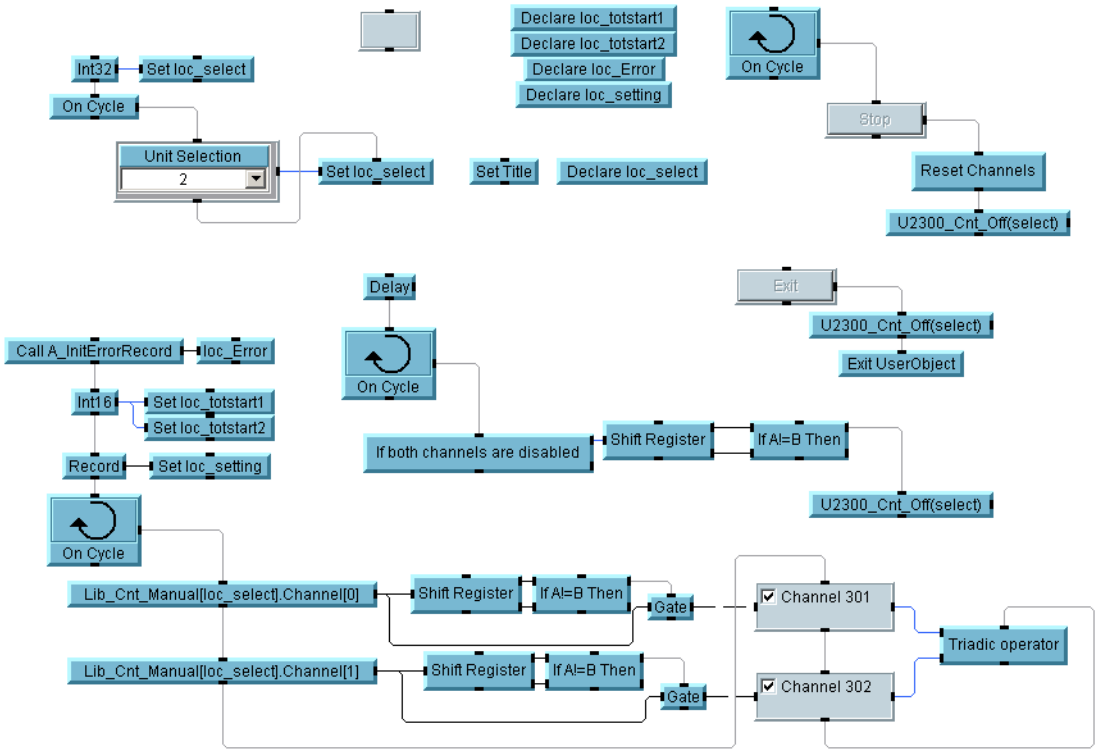
The "U2300_CNT.vee" is a standalone application that provides control for the counter of the U2300A Series DAQ devices. The user can control up to a maximum of six units of DAQ within the cardcage or via any USB port. The application will automatically detect the DAQ when it is connected to the pc via USB port. The main workspace is as shown below.

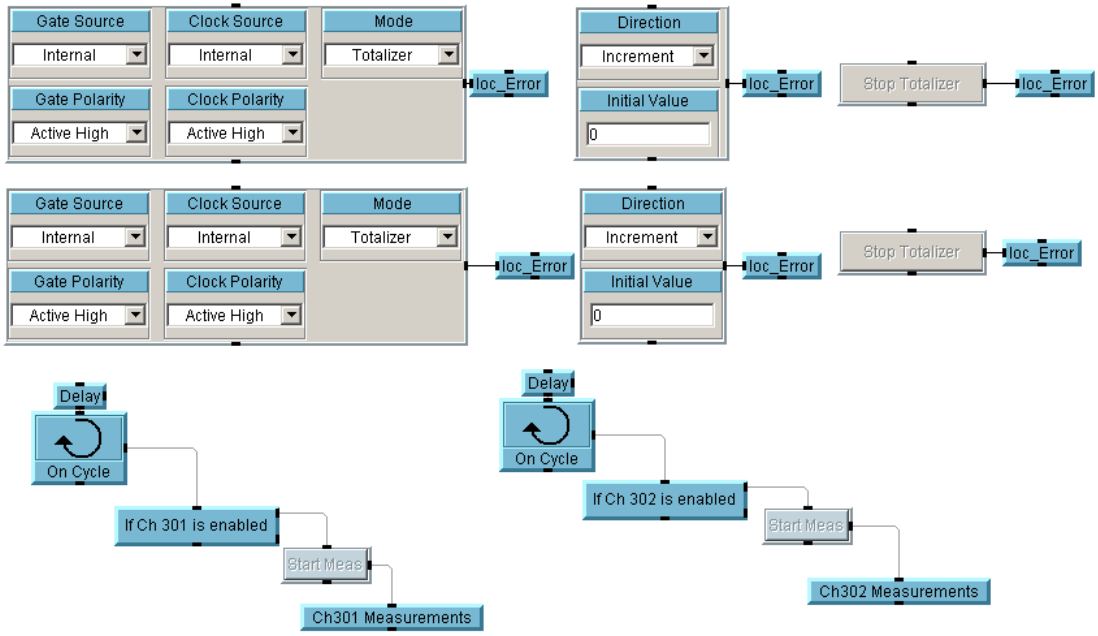


Users have control over the gate source and polarity, clock source and polarity, the ability to measure the totalizer, frequency, period and pulse width.

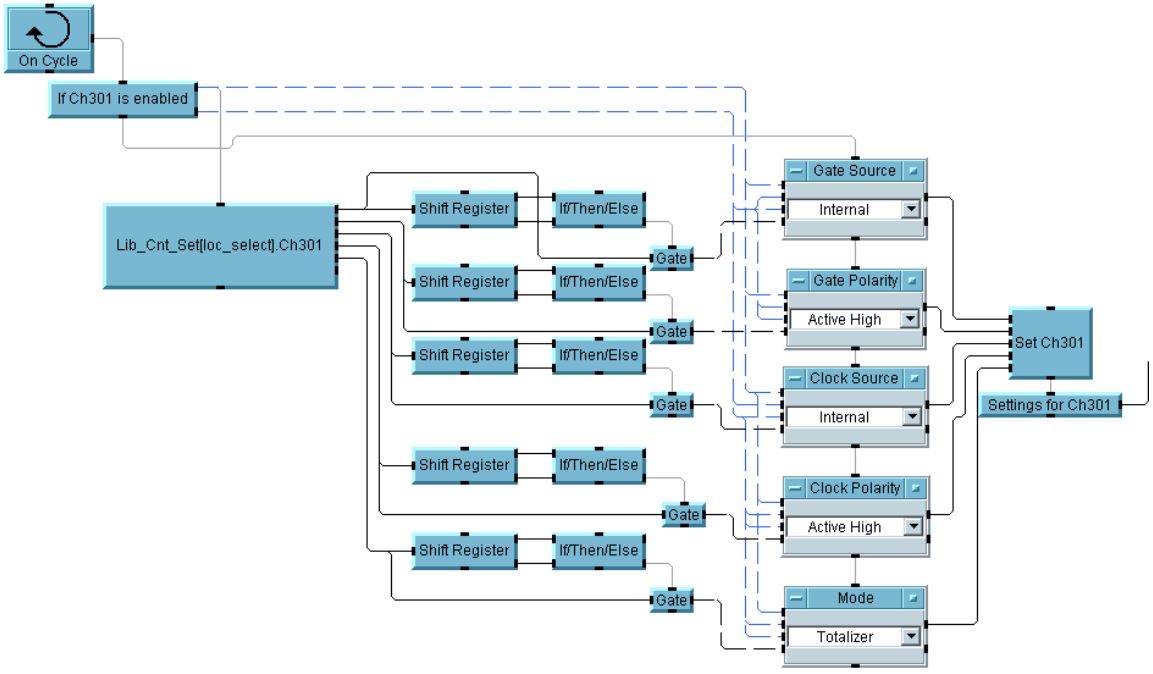

Counter control
Model : U2351A
Serial : TW46400017
Unit Selection
2
Channel 301
Gate Source
Internal
Clock Source
Internal
Mode
Totalizer
Direction
Increment
Totalizer
56106332
Gate Polarity
Active High
Clock Polarity
Active High
Initial Value
0
Frequency (kHz)
Period (ms)
Start Meas
Pulse Width (ms)
Stop Totalizer
Channel 302
Gate Source
Internal
Clock Source
Internal
Mode
Totalizer
Direction
Increment
Totalizer
Gate Polarity
Active High
Clock Polarity
Active High
Initial Value
0
Frequency (kHz)
Period (ms)
Start Meas
Pulse Width (ms)
Stop Totalizer
Stop
Exit
Counter control
Model : U2351A
Serial : TW46400017
Unit Selection
2
Channel 301
Gate Source
Internal
Clock Source
External
Mode
Measurement
Direction
Increment
Totalizer
Gate Polarity
Active High
Clock Polarity
Active High
Initial Value
0
Frequency (kHz)
5.002
Period (ms)
0.199
Start Meas
Pulse Width (ms)
0.099
Stop Totalizer
Channel 302
Gate Source
Internal
Clock Source
Internal
Mode
Totalizer
Direction
Increment
Totalizer
Gate Polarity
Active High
Clock Polarity
Active High
Initial Value
0
Frequency (kHz)
Period (ms)
Start Meas
Pulse Width (ms)
Stop Totalizer
Stop
Exit

The detailed view of the control is made up from several user-objects running in separate threads. The unit selection control allows the user to choose the DAQ unit they want to control. This will set the "loc_select" variable. Once there is a changed in this variable, the controls will be directed to that particular DAQ.

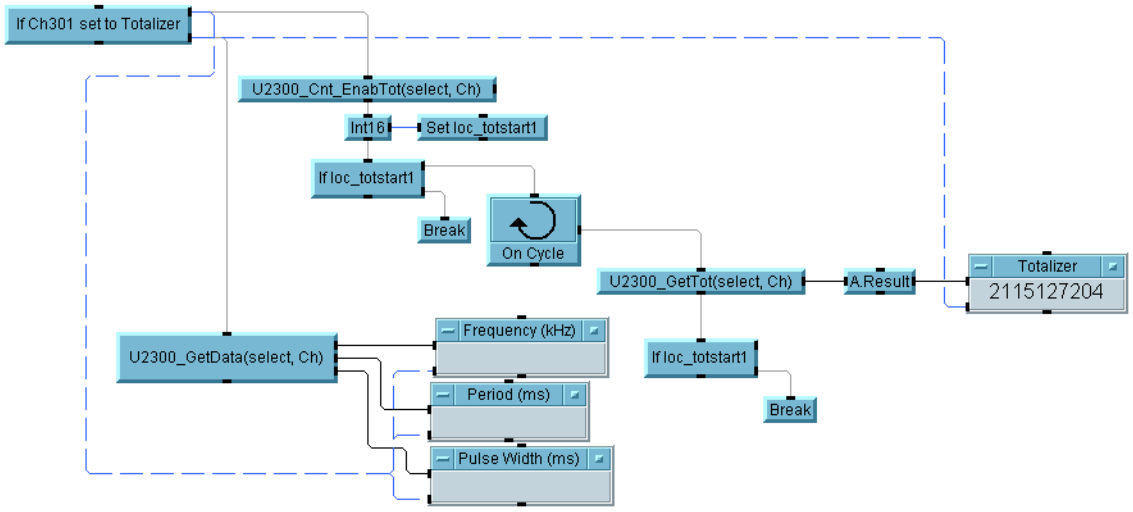



Gate Source
Internal
Clock Source
Internal
Mode
Totalizer
Gate Polarity
Active High
Clock Polarity
Active High
loc_Error
Direction
Increment
Initial Value
0
loc_Error
Stop Totalizer
loc_Error
Gate Source
Internal
Clock Source
Internal
Mode
Totalizer
Gate Polarity
Active High
Clock Polarity
Active High
loc_Error
Direction
Increment
Initial Value
0
loc_Error
Stop Totalizer
loc_Error
Delay
On Cycle
If Ch 301 is enabled
Start Meas
Ch301 Measurements
Delay
On Cycle
If Ch 302 is enabled
Start Meas
Ch302 Measurements

Channel 301 counter settings are controlled by another user-object called "Ch301 controls". Within this user-object, it is running on an individual loop, it will update the variable and update the DAQ once there is a change in the setting.



The measurement portion is made up of another user-object on a separate thread.


If Ch301 set to Totalizer
U2300_Cnt_EnabTot(select, Ch)
Int16
Set loc_totstart1
If loc_totstart1
Break
On Cycle
U2300_GetTot(select, Ch)
A.Result
Totalizer
2115127204
U2300_GetData(select, Ch)
Frequency (kHz)
Period (ms)
Pulse Width (ms)
If loc_totstart1
Break

A list of all the userfunctions in the "U2300_CNT.vee" application is as given below.

No	Userfunction	Description
1	A_InitErrorRecord ()	Initializes the error handling variable record (loc_Error). Used for the functions that only sets the DAQ.
2	A_InitOutputRecord ()	Initializes the error handling variable of 6 records (loc_Output). Used for the functions that queries the DAQ.
3	A_InitSngOutputRecord()	Initializes the error handling variable of 1 record (loc_Output). Used for the functions that queries the DAQ.
4	EnumerateVisaAddresses ()	Function that uses "visa32.dll" to detect all instruments that are connected to the pc.
5	HandleVisaReturnCode ()	Function that handles error codes generated by "visa32.dll" while searching for instruments
6	InstrAutoDetect ()	Function that automatically detects any instrument that is connected to the pc via GPIB, USB or LAN interface.
7	InstrInterfaceType (str)	Function that distinguish between a GPIB or USB interface.
8	InstSearch_MsgBox ()	Function that displays a message "Please wait... Searching for instrument"
9	PopUpMesgBox (mesg)	Multipurpose display message box.
10	PopUpMesgBoxDec (mesg)	Display message box with a "YES" and "NO" button.
11	String_IndexChar (string,char)	Finds the all the indices of the char in the string. Returns array of indices. If char doesn't exist in the string, this function returns -1. Limited to 100 indices
12	U2300_Cnt_ClkPol (Select, Polarity, Channel)	Sets the counter's input clock polarity for the given channel and selected DAQ.
13	U2300_Cnt_ClkSour (Select, Source, Channel)	Sets the counter's input clock source for the given channel and selected DAQ.
14	U2300_Cnt_EnabTot (Select, Channel)	Enables the counter's totalizer for the given channel and selected DAQ.
15	U2300_Cnt_GatePol (Select, Polarity, Channel)	Sets the counter's input gate polarity for the given channel and selected DAQ.
16	U2300_Cnt_GateSour (Select, Source, Channel)	Sets the counter's input gate source for the given channel and selected DAQ.

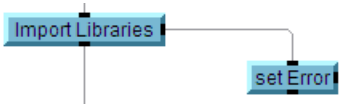
17	U2300_Cnt_Init ()	This function must be called first to establish communication with the U2300 unit. It obtains the model and serial number.
18	U2300_Cnt_InitVar ()	This function initializes all the variables used in this program.
19	U2300_Cnt_Manual ()	This function holds the main controls to the counter application.
20	U2300_Cnt_Off (Select)	Function to abort counter measurements on selected DAQ.
21	U2300_Cnt_SetTot (Select, Channel)	Start totalizer measurement for the given channel and selected DAQ.
22	U2300_Cnt_SetTotDir (Select, Direction, Channel)	Sets the direction of the totalizer for the given channel and selected DAQ.
23	U2300_Cnt_SetTotIVal (Select, Value, Channel)	Sets the initial value of the totalizer for the given channel and selected DAQ.
24	U2300_CntStopTot (Select, Ch)	Stop totalizer measurement for the given channel and selected DAQ.
25	U2300_GetFreq (Select, Channel)	Get the frequency measurement for the given channel and selected DAQ.
26	U2300_GetModel ()	Get the model of the attached U2300.
27	U2300_GetPeriod (Select, Channel)	Get the period measurement for the given channel and selected DAQ.
28	U2300_GetPWidth (Select, Channel)	Get the pulse width measurement for the given channel and selected DAQ.
29	U2300_GetSerial ()	Get the serial number of the attached U2300.
30	U2300_GetTot (Select, Channel)	Get the totalizer value for the given channel and selected DAQ.
31	U2300_Init ()	Perform a "Reset" and "Clear" the event registry and error queues of the U2300. Dynamically sets the interface address.

Temperature Monitoring Application (U2300_TempMonitor.vee)

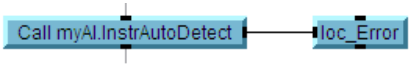
This application continuously monitor up to six analog input channels, mainly channel 101 to channel 106. The polarity of these channels are fixed as bipolar and 10 V range. It can support any U2300A Series models. The DAQ can either be on a USB port or in a cardcage. However, the present program can only support one DAQ at a time. The application uses the temperature coefficients for LM19 and LM60 temperature sensor from National Semiconductor.

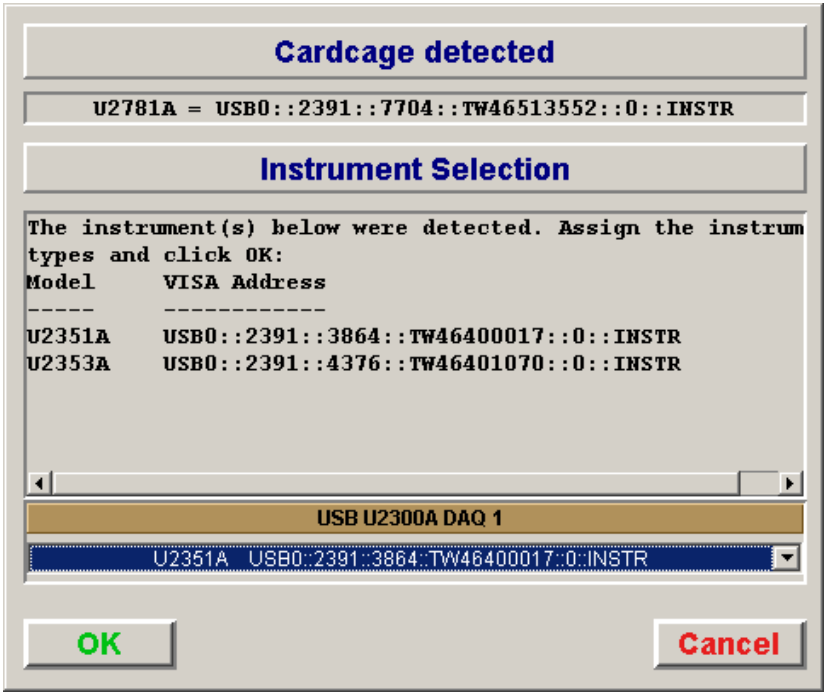
The user can also set either one or both the analog output channels (201 and 202) to output from 2.4 V to 10 V. The application also plots the graphs of all the six channels that can be enabled or disabled at any time. This program uses the userfunctions imported from "Simple U2300_AI.vee" and "Simple U2300_AO.vee". It does not use "U2300_AI.vee" and "U2300_AO.vee" as these programs have complex variable constants that have the same declaration.

The application starts by importing in the userfunctions from the two VEE files. The object is located in "Main" workspace called "Import Libraries"

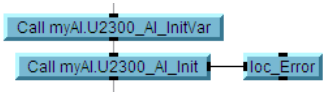


Once the userfunctions are imported, it will do an auto-detection for any instrument that is connected to the computer. The function "InstrAutoDetect ()" will display the identification and the addresses of the instrument(s) and prompt the user to select it.

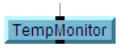


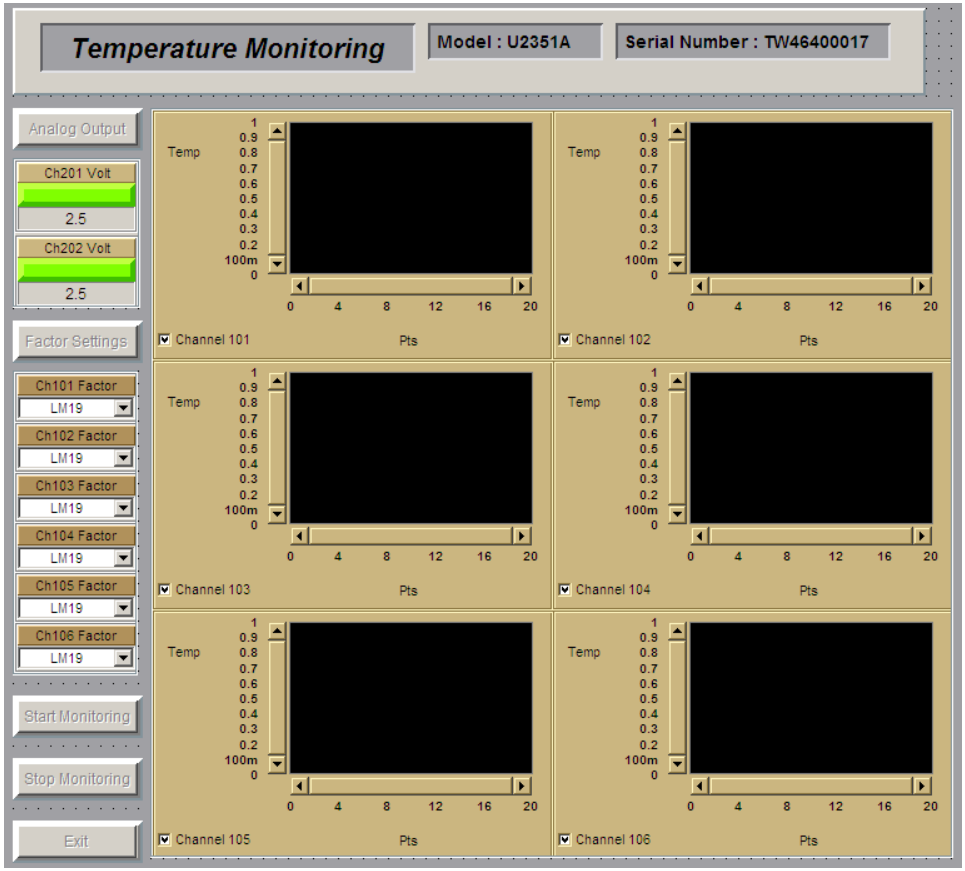


If no instrument is detected, it will give an error message and then terminate the program. The following action is used to initialize all the variables used in the program, start communicating with the instrument and obtain its model and serial number. It also determines the maximum sampling rate that is settable depending on model.



The main collection of functions that controls the temperature monitoring are contain within the object called "TempMonitor".

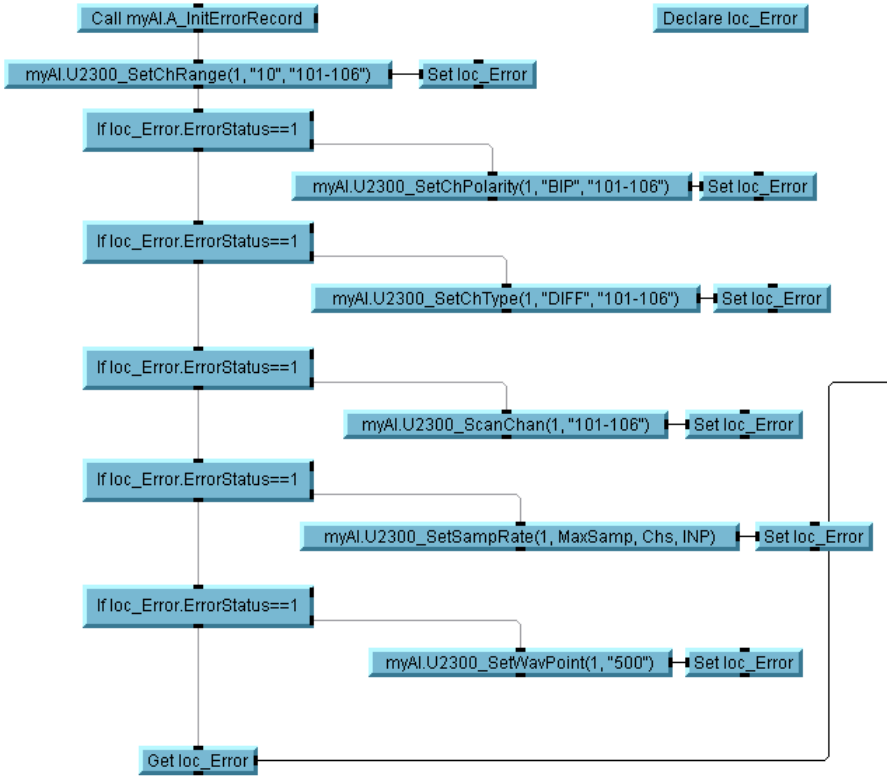




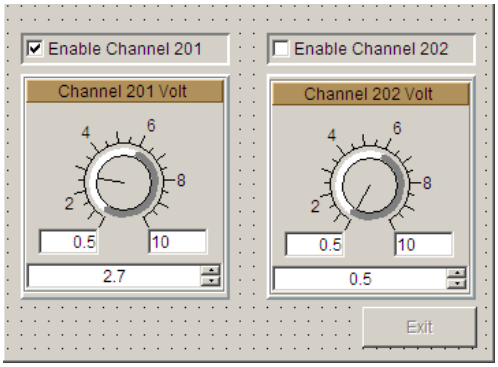
When the temperature monitoring is activated, the user can see the above screen. User is able to view the six channels, set the analog output voltage, set the temperature coefficient factors, select the temperature sensor that is connected to each channel and activate the monitoring itself.

Before any monitoring can be initiated, the channels have to be properly configured. The object "Setup for measurement" makes use of functions from "Simple U2300_AI.vee". The functions are "U2300_SetChRanges (Select, Range, Channels)", "U2300_SetChPolarity (Select, Polarity, Channels)", "U2300_SetChType (Select, Type, Channels)",

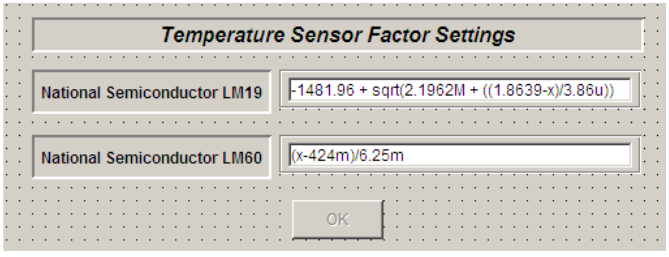
"U2300_ScanChan (Select, Channels)", "U2300_SetSampRate (Select, Maximum Rate, Channels, Rate)" and "U2300_SetWavPoint (Select, No of Points)".

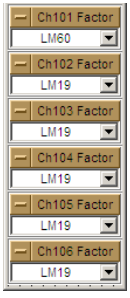


The "Analog Output" button allows the user to set the output voltage for channel 201 and 202. It makes use of functions imported from "Simple U2300_AO.vee" like "U2300_SetVolt (Select, Volt, Channel)", "U2300_AO_Off (Select)" and "U2300_GetError (Select)". These functions are used to set the voltage of the particular channel, switch it off and also read any errors generated by the U2300A units.

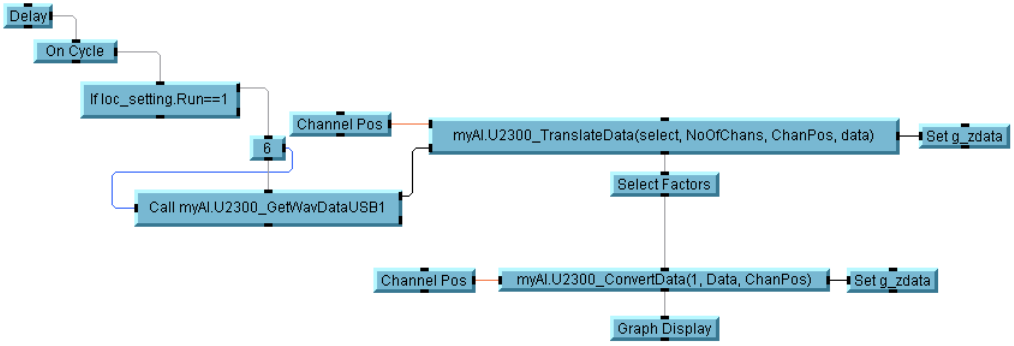


Users can change the factor settings for the temperature sensor via the "Factor Settings" button. Currently the sensor factors that are programmed in are of LM19 and LM60 from National Semiconductor. Users can also determine the sensors that are connected to each respective channel.





Functions that are used in getting the waveform, converting it from binary to decimal and applying the coefficient factors are "U2300_GetWavData (Select)", "U2300_TranslateData (Select, Chan#, Channel Position, Data)" and "U2300_ConvertData (Select, Data, Channel Position)" from the imported userfunctions from "Simple U2300_AI.vee".



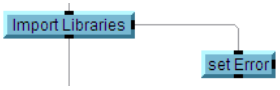
To start and stop the monitoring, the "U2300_RunUSB1 ()" and "U2300_StopUSB1 ()" functions are called. These are accessible via the "OK" buttons on the userobjects called "Start Monitoring" and "Stop Monitoring" respectively. These functions are from the file "Simple U2300_AI.vee". The "Exit" user-object calls the "U2300_StopUSB1 ()" function while the application is still monitoring before it ensures that the analog outputs are switched off using the function "U2300_AO_Off (Select)" from "Simple U2300_AO.vee".

Simple Test Application (U2300 Auto Prog Tool.vee)

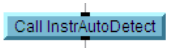
This application allows the user to test three devices via the analog input of the U2300A Series DAQ devices. The program also allows the user to control a SCPI-based power supply. The DAQ can either be on a USB port or in a cardcage. However, the present program can only support one DAQ at a time. Digital outputs from the U2300A are used to supposedly switch the DUTs. All the test results will be displayed and also the user can determine the sequence of the test, settings on the power supply and also the test limits.

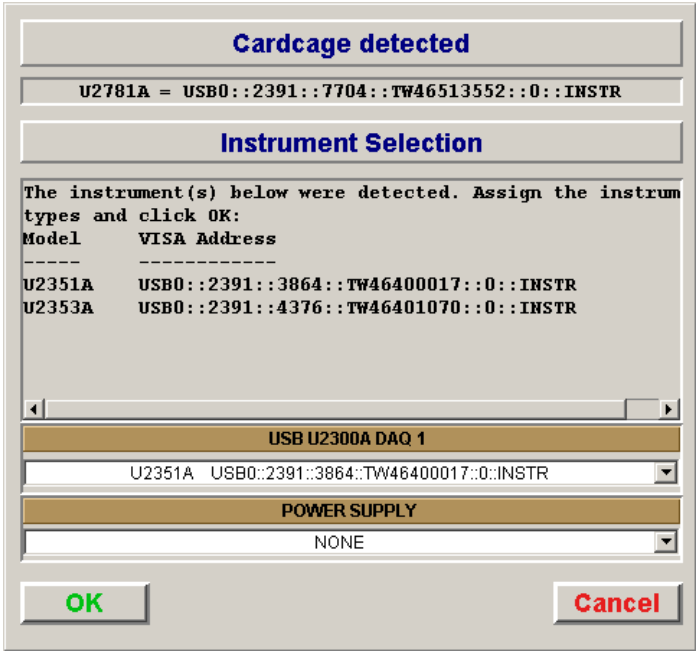
The analog input channels used are channel 101, 102 and 103. Polarity of these channels are fixed as bipolar and 10 V ranges which is the default settings. It can support any U2300A models. The digital output channel used is channel 501 with only three bits being utilized, mainly Bit 0,1 and 2 for each DUT. The application uses the userfunctions imported from "Simple U2300_AI.vee" and "Simple U2300_DIO.vee". It does not use "U2300_AI.vee" and "U2300_DIO.vee" as these programs have complex variable constant that have the same declaration.

The application start by importing in the userfunctions from the two VEE files. The object is located in "Main" workspace called "Import Libraries".



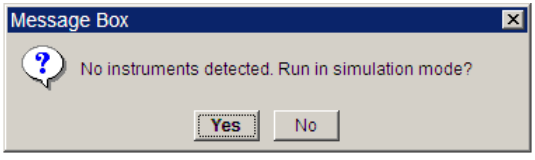
Once the userfunctions are imported, it does an auto-detection for any instrument that is connected to the computer. The function "InstrAutoDetect ()" will display the identification and the address of the detected instrument and prompt the user to select it.



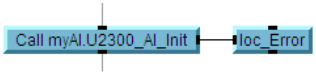


After that step, if no instruments can be detected, it will prompt the user to run in simulation mode or quit the program.

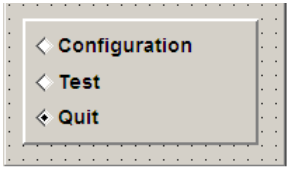
Since the emphasis on this application is on the U2300A, the user can still proceed to use it even a programmable power supply is not present. Put the selection for power supply as "NONE".



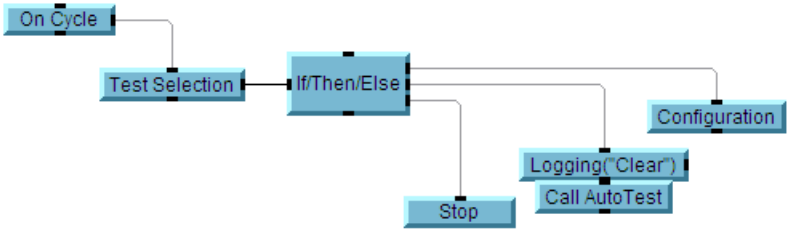
The following object is used to start communicating with the instrument and obtain its model and serial number. It also determines the maximum sampling rate that is settable depending on model.



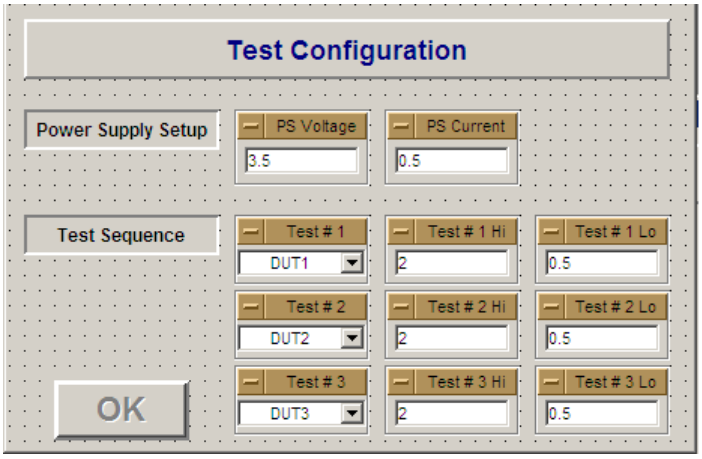
The following step will request the user to select and configure the tests or proceed to test.



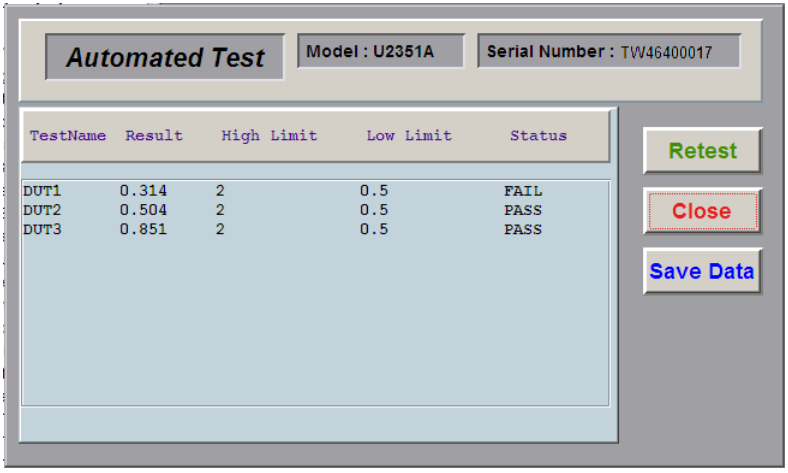
The test selection will branch out to an "If/Then/Else" object to determine which object to call based on the users selection.

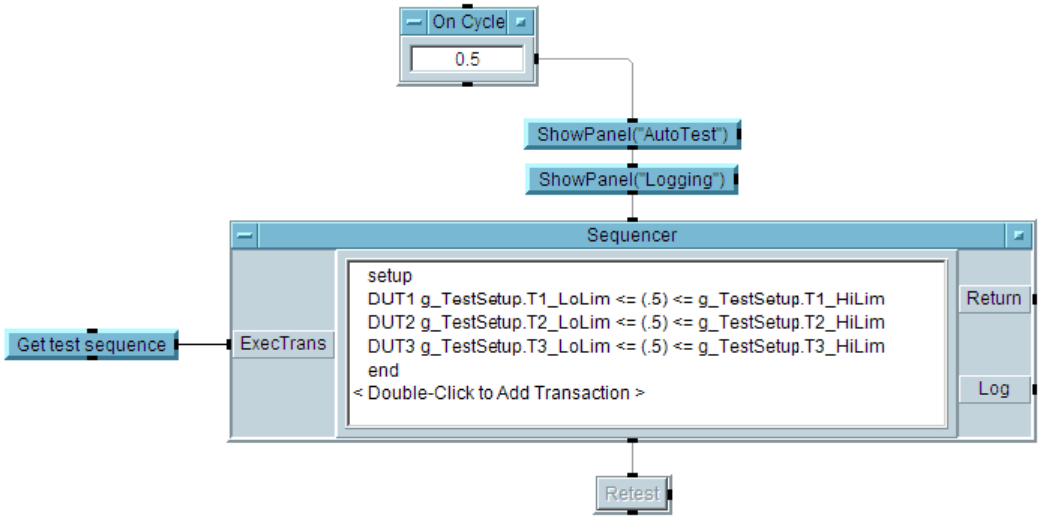


Within the configuration screen, the user can determine the voltage and current setting of the power supply. They can also set the sequence of the tests and the limits.



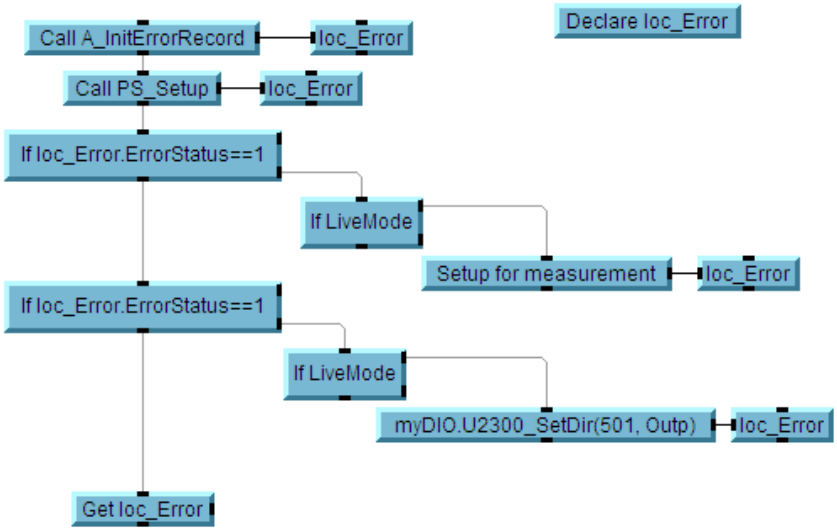
The "AutoTest" userobject uses a sequencer to handle all the tests and it passes the results to a userfunction called "Logging ()". Prior to running the "AutoTest" object, the test log is cleared. The "AutoTest" graphical user interface (GUI) allows the user to retest and save the data to a text file. It also shows the model and serial number of the DAQ currently attached to the PC.



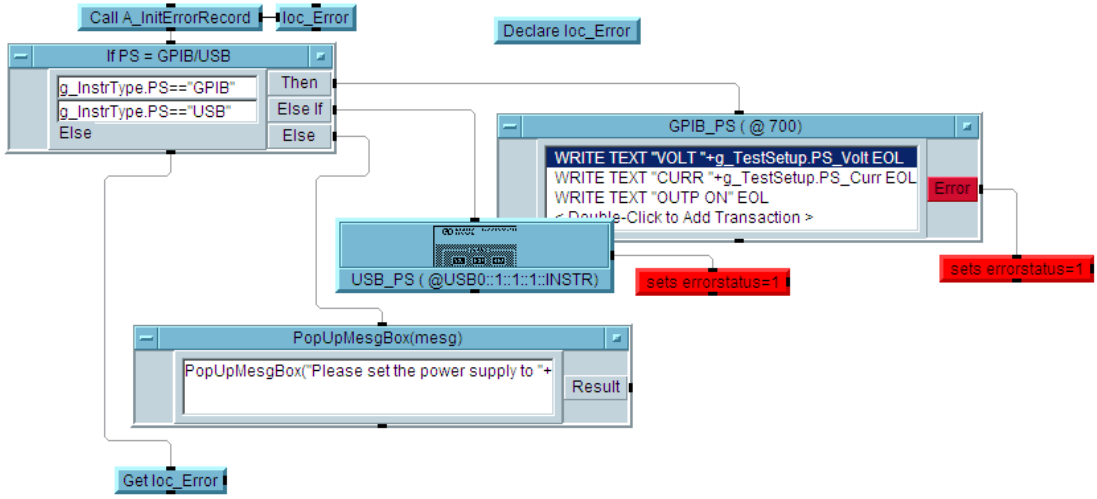


Before any test starts, the "setup" routine is called to set up the power supply. When all the tests is completed, the "end" sequence will ensure that the power supply is switched off. The sequence of testing for DUT1 to DUT3 is determined by the user.

All the test results are passed to a userfunction called "Logging ()". The "setup" routine calls the "T_Start ()" userfunction that prepares all the instruments for measurement.

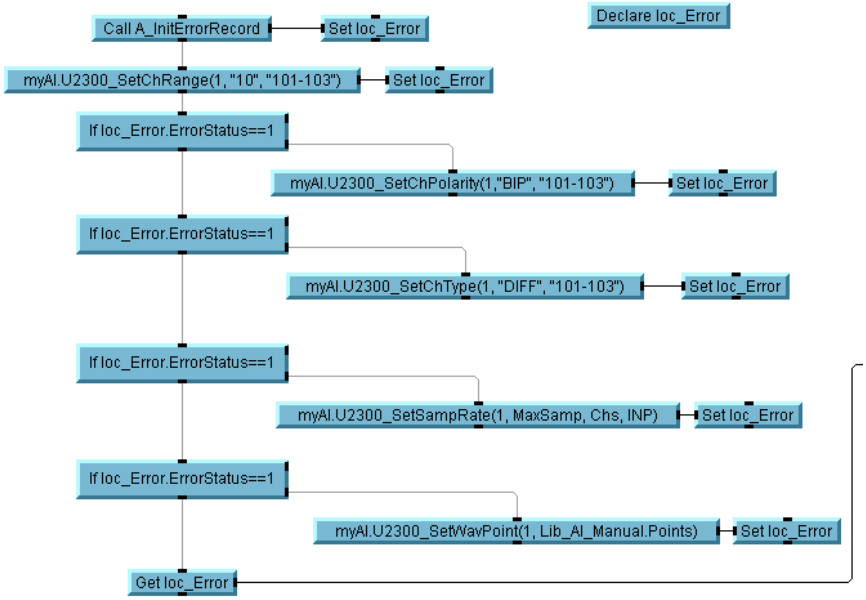


Within the "T_Start ()" userfunction, it calls upon the "PS_Setup ()" userfunction that setup the power supply. It also determines that if it is not operating in simulation mode, it will setup the DAQ for analog input in "Setup in measurement" object and the digital port "501" for output mode. "U2300_SetDir (Select, Channel, Mode)" function from "Simple U2300_DIO.vee" is use to configure channel 501 to output mode.

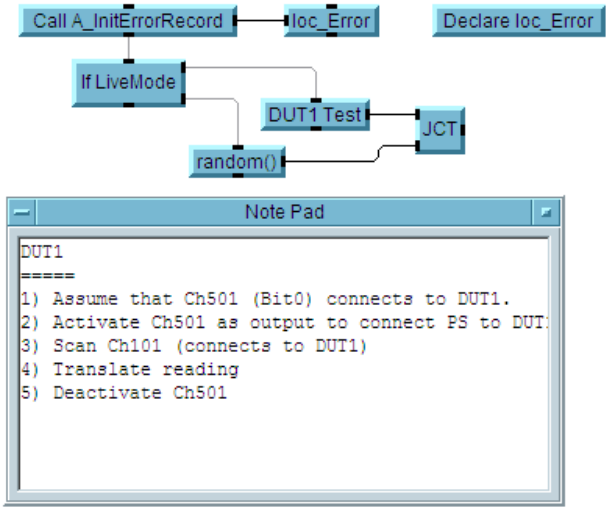


In the "PS_Setup ()" userfunction, the program determines which interface to use. In this case, only "GPIB" or "USB" based power supply is considered. If the power supply is other than the covered interfaces or without any interfaces, a message box will appear instructing the user what to do manually. The program also assumes that any programmable power supply used will have to be SCPI- based.

Before any monitoring can be initiated, the channels have to be properly configured. The object "Setup for measurement" makes use of the functions from "Simple U2300_AI.vee". The functions are "U2300_SetChRanges (Select, Range, Channels)", "U2300_SetChPolarity (Select, Polarity, Channels)", "U2300_SetChType (Select, Type, Channels)", "U2300_ScanChan (Select, Channels)", "U2300_SetSampRate (Select, Maximum Rate, Channels, Rate)" and "U2300_SetWavPoint (Select, No of Points)".

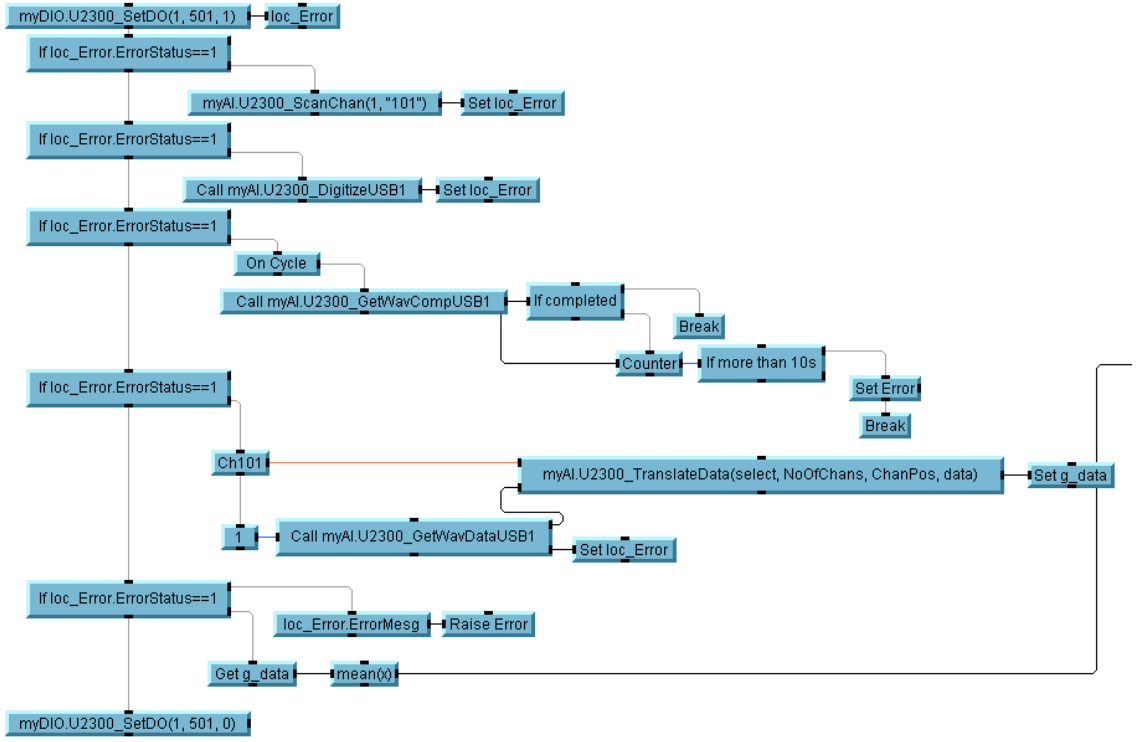

Declare Ioc_Error
Call A_InitErrorRecord
Set Ioc_Error
myAI.U2300_SetChRange(1,"10","101-103")
Set Ioc_Error
If Ioc_Error.ErrorStatus==1
myAI.U2300_SetChPolarity(1,"BIP","101-103")
Set Ioc_Error
If Ioc_Error.ErrorStatus==1
myAI.U2300_SetChType(1,"DIFF","101-103")
Set Ioc_Error
If Ioc_Error.ErrorStatus==1
myAI.U2300_SetSampRate(1, MaxSamp, Chs, INP)
Set Ioc_Error
If Ioc_Error.ErrorStatus==1
myAI.U2300_SetWavPoint(1, Lib_AI_Manual.Points)
Set Ioc_Error
Get Ioc_Error

Test function "T_DUT1 ()" checks if it is live mode before actually running the measurement, else it will generate a random number for the simulation mode.



Inside the "DUT1 Test" object, it calls upon userfunctions from "Simple U2300_AI.vee" and "Simple U2300_DIO.vee". It will set digital channel "501" bit 0 to high to signify that a connection is made between the device. Then it will scan channel "101" and set it to digitize mode for a single shot mode. This is made by the userfunctions "U2300_ScanChan (Select, Channel)" and "U2300_DigitizeUSB1 ()". The program will then monitor for the completion of the signal via "U2300_GetWavCompUSB1 ()".

Once the data is available, the userfunction "U2300_GetWavDataUSB1 ()" will read out the data in binary format and "U2300_TranslateData (Select, No of channels, Channel number, Data)" will convert the data to numerical format. It will then output the average reading and disconnect the DUT by setting bit 0 of channel 501 to low.



Test functions "T_DUT2 ()" and "T_DUT3 ()" are similar to "T_DUT1 ()" except it controls different channels. "T_End ()" is called to complete the tests by switching off the power supply and resetting the digital output of the DAQ.

Appendix

User-Defined Functions/Libraries

VEE provides 19 categories of built-in functions you can use in programs. When one of these built-in functions is not exactly right for your program, you can define your own function.

This chapter describes how to create custom functions with/using UserFunctions.

VEE Pro supports three kinds of user-defined functions:

- UserFunctions
- Compiled Functions
- Remote Functions

This chapter describes UserFunctions, Compiled Functions, and Remote Functions, in the following sections:

- About UserFunctions
- Using a Library of Functions
- About Compiled Functions
- About Remote Functions

About UserFunctions

A UserFunction is specifically designed for creating a user-defined function. You create a UserFunction by selecting it from the Device menu or by converting existing objects or an existing UserObject into a UserFunction. This section describes how to create a UserFunction. The next section describes how to convert a UserObject into a UserFunction.

To create a UserFunction, click **Device ⇒UserFunction**. An empty UserFunction window appears in the work area. Create your function by adding terminals and objects as needed. Change the name to whatever you want (spaces not allowed). See the *VEE User's Guide* or **How Do I** in *VEE Online Help* for additional details.

When the UserFunction is complete, you can iconify it or close it to get it out of the way of the rest of your program. You can call your UserFunction using a Call object in your program (**Device ⇒Call**) or other objects identified below. A UserFunction can be saved in a library and imported into a program with the Import Library object.

The advantage of creating a UserFunction over using a UserObject is that you can call a single UserFunction several times in your program. Thus, there is only one UserFunction to edit and maintain, rather than several instances of a UserObject.

When executed in `VEE 4`, or higher Execution Mode, a UserFunction will time-slice when called from Call, Formula, If/Then/Else, or Sequencer objects (only from the Function field).

A UserFunction will not time-slice when called from a To File, To String, or similar object or if the Formula object's formula is supplied via a control pin.

Converting Between UserObjects and UserFunctions

To convert a UserObject into a UserFunction, select Make UserFunction from the UserObject's object menu. The UserObject window is replaced by a UserFunction window with the same input and output terminals. The UserObject object is replaced by a UserFunction Call object.

To reconvert the UserFunction back into a UserObject, select Make UserObject from the object menu of the UserFunction window. Any calls to the UserFunction remain (you will have to manually delete them), but the UserFunction is automatically converted into a UserObject.

Calling a UserFunction from an Expression

You can call a UserFunction from an expression in a Formula object or from any expression evaluated at run time, such as from a ToFile object. The program in Figure 1 demonstrates several ways to call a UserFunction.

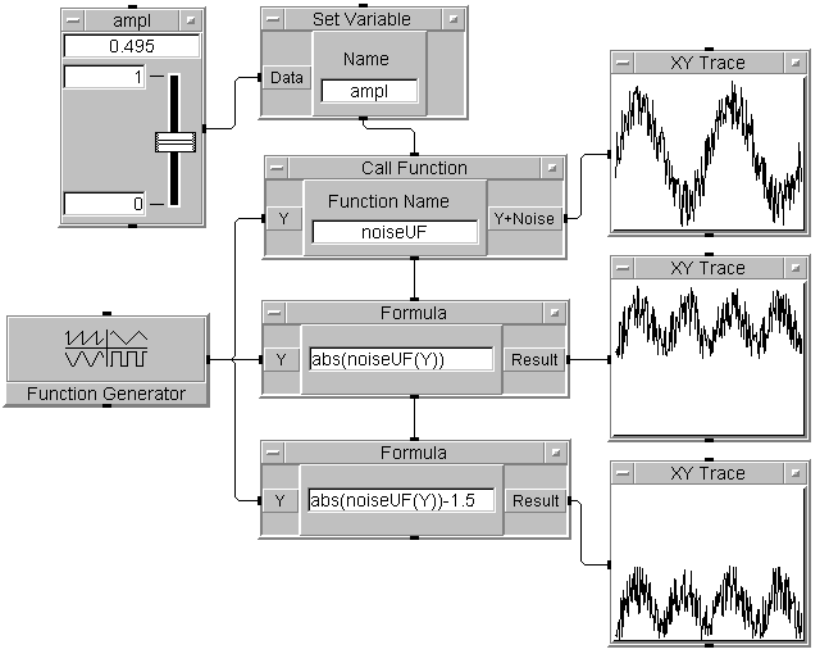


Figure 1 Calling a UserFunction from Expressions

In the program, the Call object calls the UserFunction **noiseUF** and returns a sine wave with an added noise component. The expression **abs(noiseUF(Y))** in the first Formula object returns the absolute value of the waveform returned by the UserFunction. Thus, the displayed noisy sine wave is rectified in the positive direction.

The expression **abs(noiseUF(Y))-1.5** in the second Formula object also calls the UserFunction but adds a negative dc offset to the waveform. The sequence pins are used to ensure correct propagation because the UserFunction uses the global variable.

This program is saved in the file **manual43.vee** in the examples directory.

Using a Library of Functions

Methods for creating each type of user-defined function and using it in a VEE program are similar. All these functions are called using the Call object or from certain expressions, such as in Sequencer or Formula objects. You can use any of the three kinds of user-defined functions in a library. To use a library of functions, follow these steps:

1. Import the library.

   Use the **Device ⇒Import Library** object. Select the Library Type (UserFunction, Compiled Function, or Remote Function) and fill in the appropriate fields. Specific information about these fields is explained in the associated section in this chapter.

2. Call one or more functions that are contained in the library.

   Use the Call, Formula, or Sequencer objects from the Device menu. You can also use other objects that call expressions at run time, such as If/Then/Else or To File. If you want to have multiple values returned from the function, you must use a Call object.

3. Delete the library.

   If memory management or program execution speed is a concern, use the **Device ⇒Delete Library** object to programmatically free the library from memory. Otherwise, libraries are automatically deleted when VEE exits.

Specific information about using different kinds of libraries is listed in the following sections.

The ability to call a UserFunction from an expression is very useful — especially when you include such an expression in a transaction in the Sequencer object. See *Chapter 14 (VEE Pro Advanced Techniques)* for more information about this topic.

Creating a UserFunction Library

So far we have looked at local UserFunctions that are created and used within the same program. You can also create a library of multiple UserFunctions stored externally and later imported into a program.

To create a library of UserFunctions, create the UserFunctions in the empty VEE work area and save them to a file. For example, to create a library of two UserFunctions, myRand1 and myRand2 (which add random numbers to an input value), create the two UserFunctions as shown in Figure 2.

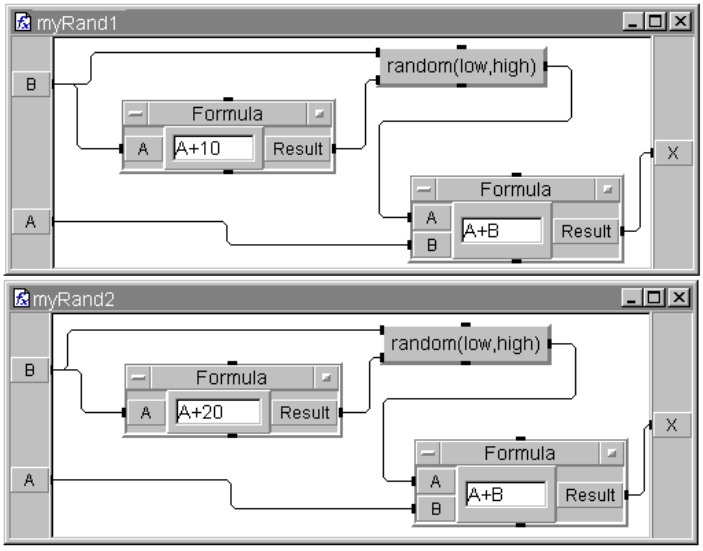


Figure 2 Creating UserFunctions for a Library

To create a UserFunction library, save the program with a name that identifies it as a library. For example, use a .vlb extension instead of .vee.

NOTE Normally, the program should contain only UserFunctions. If other objects are in the program (e.g., in Main), they are ignored when the library is imported. If you use Declare Variable objects, put them in one of the UserFunctions, not in the Main window of the library.

Importing and Calling a UserFunction

To import the UserFunction library into a program, use the Import Library object. The program in Figure 3 imports the library from the file user_func_lib and calls the UserFunctions myRand1 and myRand2.

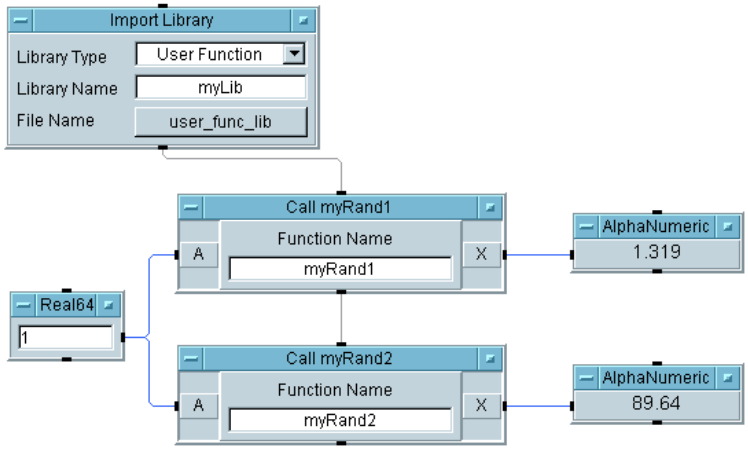


Figure 3 Importing a UserFunction Library

The Import Library object allows you to specify a type of library: User Function, Compiled Function, or Remote Function. If you select UserFunction, you also specify a Library Name and File Name.

The Library Name field specifies a local name for the library within the program. This makes it possible for the Delete Library object to delete the library from the program. In this case, Import Library attaches the name myLib to the library imported from the file user_func_lib.

The File Name field specifies the file from which to import the library, user_func_lib in this case. If you click on the File Name field you can select from a list of all library files.

This program is simple so it is not necessary to delete the UserFunction library after it is used. In a large program with calls to large libraries, deleting a library when you no longer need it reduces the program's memory requirements.

NOTE

You cannot edit UserFunctions imported with Device ⇒Import Library, but you can view their contents and set breakpoints for debugging. To view imported UserFunctions, use the Program Explorer.

You can *merge* a library of UserFunctions using File ⇒Merge Library. Once the library is merged into your program, you can edit the individual UserFunctions with Edit ⇒Edit UserFunction.

Merging UserFunctions

Merging a UserFunction lets you make it part of your program. Since it is not imported, you can modify it as needed. A merged UserFunction is saved with the VEE program and does not change if the original library changes.

To merge a UserFunction into a program, select **File ⇒Merge Library**. A dialog box opens displaying the files in the library directory. Select the file containing the UserFunction library you want and click `Open`.

About Compiled Functions

A Compiled Function is created by dynamically linking a library written in C, C++, FORTRAN, or Pascal, to the VEE process. A library of compiled functions is called a dynamic link library (DLL) in Microsoft Windows.

Creating a Compiled Function is considerably more difficult than creating a UserFunction. Once you have written a library of functions in C or another language, you will need to compile the functions into a DLL or shared library. You will also have to create a definition file that will provide VEE with information it needs to call your function.

Using a Compiled Function

To use a Compiled Function, you:

1. Write the external program.
2. Create the DLL and a definition file.
3. Import the library and call the function from VEE.
4. Delete the library from VEE memory when you are done.

The methods for importing a Compiled Function library and for calling the function are very similar to those for UserFunction libraries. The Import Library object attaches the DLL to the VEE process and parses the definition file declarations.

The definition file defines the type of data passed between the external library and VEE. (This file is discussed later in this section.) The Compiled Function can then be called with the Call object or from such objects as Formula and If/Then/Else.

Design Considerations for Compiled Functions

Using Compiled Functions, you can develop time-sensitive routines in another language and integrate them directly into your VEE program. You can also use Compiled Functions to keep proprietary routines secure.

Because Compiled Functions do not timeslice (i.e., they execute until they are done without interruption) they are only useful for specific purposes that are not otherwise available in VEE.

You can extend the capabilities of your VEE program by using Compiled Functions, but it adds complexity to the VEE process. *The key design goals should be:*

- *Keep the purpose of the external routine highly focused on a specific task*
- *Use Compiled Functions only when the capability or performance you need is not available using a VEE UserFunction or an* Execute Program *escape to the operating system.*

You can use any operating system facilities available in the program to be linked, including math routines, instrument I/O, etc. *However, you cannot access any VEE internal functions from within the external program to be linked.*

Although the use of Compiled Functions provides enhanced VEE capabilities, some problems can occur. A few key ones are:

- VEE cannot trap errors originating in the external routine. Because your external routine becomes part of the VEE process, any errors in that routine propagate back to VEE. A failure in the external routine may cause VEE to "hang" or otherwise fail. You need to be sure of what you want the external routine to do and provide for error checking in the routine. If your external routine exits so will VEE.
- Your routine must manage all memory that it needs. Be sure to deallocate any memory that you may have allocated when the routine was running.
- Your external routine cannot convert data types the way VEE does. You should configure the data input terminals of the Call object to accept *only* the type and shape of data that is compatible with the external routine.

- If your external routine accepts arrays, it must have a valid pointer for the type of data it will examine. The routine also must check the size of the array on which it is working. The best way to do this is to pass the size of the array from VEE as an input to the routine, separate from the array itself. If your routine overwrites values of an array passed to it, use the return value of the function to indicate how many of the array elements are valid.
- System I/O resources may become locked. Your external routine is responsible for timeout provisions, etc.
- If your external routine performs an invalid operation, such as overwriting memory beyond the end of an array or dereferencing a null or bad pointer, this can cause VEE to exit or error with a General Protection Fault.
- If your external routine has arrays or char* parameters, the memory passed to these routines must be allocated in VEE. You should allocate this memory by doing the following:
  - For an array input, use an Alloc Array object of the appropriate type, and set the size appropriately.
  - For a string input, use a Formula object. Delete the data input terminal from the Formula object and enter an expression like 256*"a". This creates a string that is 256 characters long (plus a null byte) filled with a's. Most VXI*plug&play* functions will not write more than 256 characters into a Text parameter. However, it is best to check the Help on each function panel that requires a `Text` input to be sure.

Importing and Calling a Compiled Function

You can import a DLL into your VEE program with the **Import Library** object, then call the Compiled Function with the Call object. The process is very much like importing a library of UserFunctions and calling the functions, as described at the beginning of this chapter.

To import a Compiled Function library, select Compiled Function in the Library Type field.

Just as for a UserFunction, the Library Name field attaches a name to identify the library within the program, and the File Name field specifies the file from which to import the library. For a Compiled Function, there is a fourth field, which specifies the name of the Definition File, shown in Figure 4.

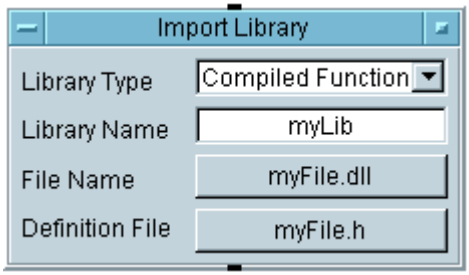


Figure 4 Using Import Library for Compiled Functions

The definition file defines the type of data passed between the external routine and VEE. It contains prototypes for the functions.

After importing the library with Import Library, you can call the Compiled Function by specifying the function name in the Call object. For example, the Call object in Figure 5 calls the Compiled Function named **myFunction**.

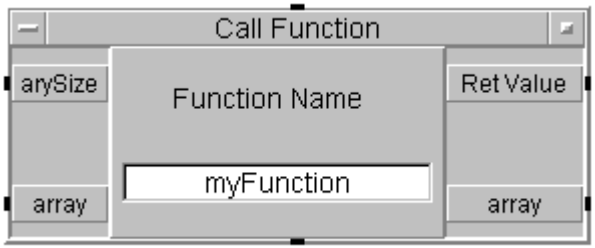


Figure 5 Using Call for Compiled Functions

Select the desired function using Select Function from the Call object menu or from the Function & Object Browser (under **Device ⇒Function & Object Browser**), or type the name in the Call object.

If VEE recognizes the function, the input and output terminals of the Call object are configured automatically for the function. (The necessary information is supplied by the definition file.) You can reconfigure the Call input and output terminals by selecting Configure Pinout in the object menu.

VEE configures the Call object with the input terminals required by the function and with a Ret Value output terminal for the return value of the function. There also will be an output terminal corresponding to each input that is passed by reference.

You can also call the Compiled Function by name from an expression in a Formula object or from other expressions evaluated at run time. For example, you could call a Compiled Function by including its name in an expression in a Sequencer or ToFile transaction.

However, only the Compiled Function's return value (Ret Value in the Call object) can be obtained from within an expression. If you want to obtain other parameters from the function, you have to use the Call object.

The Definition File

The Call object or Formula expression determines the type of data it should pass to the function based on the contents of the definition file. The definition file defines the type of data the function returns, the function name, and the arguments the function accepts. The data has the following form:

```
<return type> <function name> (<type>
<paramname>, <type> <paramname>, ...) ;
```

Where:

- `<return type>` can be: int, short, long, float, double, char*, or void.

- `<function name>` can be a string consisting of an alpha character followed by alphanumeric characters, up to a total of 512 characters.
- `<type>` can be: int, short, long, float, double, int*, char*, short*, long*, float*, double*, char**, or void.
- `<paramname>` can be a string consisting of an alpha character followed by alphanumeric characters, up to a total of 512 characters. The parameter names are optional, but recommended. If a parameter is to be passed by reference, the parameter name must be preceded by the indirection symbol (`*`).

The valid return types are:

- character strings (char*, corresponding to the VEE Text data type)
- integers (short, int, long, corresponding to the VEE Int16 and Int32 data types)
- single and double precision floating point real numbers (float and double corresponding to the VEE Real32 and Real64 data types).

If you specify "pass by reference" for a parameter by preceding the parameter name with `*`, VEE will pass the address of the information to your function. If you specify "pass by value" for a parameter by leaving out the `*`, VEE will copy the value (rather than the address of the value) to your function. You will want to pass the data by reference if your external routine changes that data for propagation back to VEE. *All arrays must be passed by reference.*

Any parameter passed to a Compiled Function by reference is available as an output terminal on the Call object. The output terminals will be Ret Value for the function's return value, plus an output for each input parameter that was passed by reference.

VEE pushes 144 bytes on the stack. This allows up to 36 parameters to be passed by reference to a Compiled Function. Up to 36 long integer parameters or 18 double-precision floating-point parameters may be passed by value.

VEE allows both "enclosed" comments and "to-end-of-line" comments in the definition file.

"Enclosed" comments use the delimiter sequence /*comments*/, where /* and */ mark the beginning and end of the comment, respectively.

"To-end-of-line" comments use the delimiting characters // to indicate the beginning of a comment that runs to the end of the current line.

Building a C Function

The following C function accepts a real array and adds 1 to each element in the array. The modified array is returned to VEE on the Array terminal, while the size of the array is returned on the Ret Value terminal. This function, once linked into VEE, becomes the Compiled Function called in the VEE program shown in Figure 6.

```
/*
  C code from manual49.c file
*/

#include <stdlib.h>

#ifdef WIN32
#  define DLLEXPORT __declspec(dllexport)
#else
#  define DLLEXPORT
#endif

/* The description will show up on the Program Explorer when you select
"Show Description" from the object menu and the Function Selection
dialog box in the small window on the bottom of the box.
*/
DLLEXPORT char myFunc_desc[] = "This function adds 1.0 to the array
```

```
passed in";

DLLEXPORT long myFunc(long arraySize, double *array) {
  long i;

  for (i = 0; i < arraySize; i++, array++) { *array += 1.0; }

  return(arraySize);
}
```

The definition file for this function is as follows:

```
/*
definition file for manual49.c
*/

long myFunc(long arraySize, double *array);
```

(This definition is the same as the ANSI C prototype definition in the C file.)

You must include any header files on which the routine depends. The library should link against any other system libraries needed to resolve the system functions it calls.

The example program uses the ANSI C function prototype. The function prototype declares the data types that VEE should pass into the function.

The array has been declared as a pointer variable. VEE will put the addresses of the information appearing on the Call data in terminals into this variable. The array size has been declared as a long integer. VEE will put the value (not the address) of the size of the array into this variable. The positions of both the data input terminals and the variable declarations are important. The addresses of the data items (or their values) supplied to the data input pins (from top to bottom) are placed in the variables in the function prototype from left to right.

One variable in the C function (and correspondingly, one data input terminal in the Call object) is used to indicate the size of the array. The arraySize variable is used to prevent data from being written beyond the end of the array. If you overwrite the bounds of an array, the result depends on the language you are using. In Pascal, which performs bounds checking, a run-time error will result, stopping VEE. In languages like C, where there is no bounds checking, the result will be unpredictable, but intermittent data corruption is probable.

This example has passed a pointer to the array so it is necessary to dereference the data before the information can be used.

The arraySize variable has been passed by value so it will not show up as a data output terminal. However, here we have used the function's return value to return the size of the output array to VEE. This technique is useful when you need to return an array that has fewer elements than the input array.

The program in Figure 6 calls the Compiled Function created from the example C program:

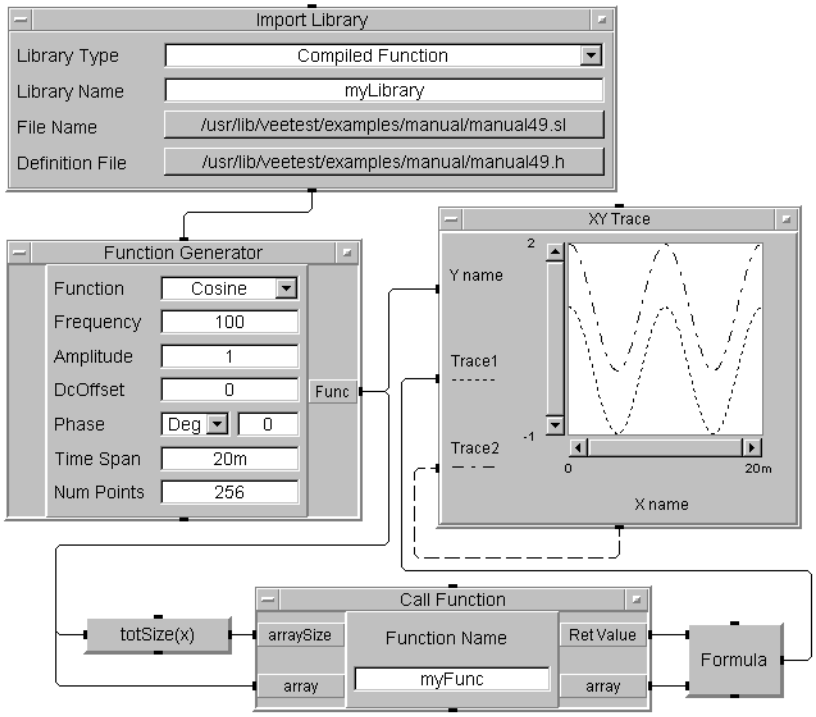


Figure 6 Program Calling a Compiled Function

The example in Figure 6 is saved in the file **manual49.vee** in the **examples** directory. The C file is saved as **manual49.c**, the definition file as **manual49.h** and the shared library as **manual49.sl**.

Creating a Dynamic Link Library

VEE provides access to DLLs through the Call object and through formula objects.

NOTE This section describes how to call a DLL, not how to write a DLL. VEE Version 3.2 and greater only calls 32-bit DLLs, not 16-bit DLLs.

Creating the DLL

Create the DLL before writing the VEE program. Create the DLL as you would any other DLL, except that only a subset of C types are allowed. (See "Creating the Definition File" on page 78.)

Declaring DLL Functions If you are using Microsoft Visual C++ Version 2.0 or greater, the function definition should be:

```
__declspec(dllexport) long myFunc (...);
```

This definition eliminates the need for a `.DEF` file to export the function from the DLL. Use the following command line to compile and link the DLL:

```
cl /DWIN32 $file.c /LD
```

`/LD` creates a DLL. Use `/Zi` to generate debug information.

The MS linker links to the C multi-threaded Runtime Library by default. If you use functions like `GetComputerName()`, you need to link against **Kernel32.lib**. The compile/link line would look like:

```
cl /DWIN32 file.c /LD /link Kernel32.lib
```

Declaring DLL Functions To work with VEE, DLL functions can be declared as `__declspec(dllexport)` using Microsoft C++ version 2.0 or greater. This eliminates the need for a `.DEF` file. For example, a generic function could be created as follows:

```
__declspec(dllexport) long genericFunc(long a) {return
(a*2); }
```

If you are not using Microsoft Visual C++, the `.DEF` file contains:

```
EXPORTS genericFunc
```

And the function definition looks like:

```
long genericFunc(long a);
```

Creating the Definition File The definition file contains a list of prototypes of the imported functions. VEE uses this file to configure the Call objects and to determine how to pass parameters to the DLL function. The format for these prototypes is:

```
<return type> <modifier> <function name> (<type> <paramname>, <type>
<paramname>, ...) ;
```

where:

- `<return type>` can be: int, short, long, float, double, char*, or void.
- `<function name>` can be a string consisting of an alpha character followed by alphanumeric characters, up to a total of 512 characters.
- `<modifier>` can be _cdecl, _pascal, or _stdcall.
- `<type>` can be: int, short, long, float double, int*, char*, short*, long*, float*, double*, char**, or void.
- `<paramname>` can be a string consisting of an alpha character followed by alphanumeric characters, up to a total of 512 characters. The parameter names are optional, but recommended. If a parameter is to be passed by reference, the parameter name must be preceded by the indirection symbol (`*`).

For example:

Pass in four parameters, return a long:

```
long aFunc(double *,long param2,long *param3,
char *);
```

No input parameters, return a double:

```
double aFunc();
```

Pass in a string, return a long:

```
long aFunc(char *aString);
```

Pass in an array of strings, return a long:

```
long aFunc(char **aString);
```

Parameter Limitations

A DLL function called from VEE pushes a maximum of 144 bytes on the stack. This limits the number of parameters used by the function. Any combination of parameters may be used as long as the 144-byte limit is not exceeded. A long

uses four bytes, a double uses eight bytes and a pointer uses four bytes. For example, a function could have 36 longs, or 18 doubles, or 20 pointers and 8 doubles.

The Import Library Object

Before you can use a Call object or Formula box to execute a DLL function you must import the function into the VEE environment via the Import Library object. On the Import Library object, select Compiled Function under Library Type. Enter the correct definition file name using the Definition File button. Finally, select the correct file using the File Name button. The Library Name button assigns a logical name to a set of functions and does not need to be changed.

The Call Object

Before using a DLL function with the Call object you must configure the Call object. The easiest way to do this is to select Load Lib on the Import Library object menu to load the DLL file into the VEE environment. Then, select Select Function on the Call object menu.

VEE will bring up a dialog box with a list of all the functions listed in the definitions file. When you select a function, VEE automatically configures the Call object with the correct input and output terminals and function name.

You can also configure the Call object manually by modifying the function name and adding the appropriate input and output terminals:

1. Configure the same number of input terminals as there are parameters passed to the function. The top input terminal is the first parameter passed to the function. The next terminal down from the top is the second parameter, etc.
2. Configure the output terminals so the parameters passed by reference appear as output terminals on the Call object. Parameters passed by value cannot be assigned as output terminals. The top output terminal is the value returned by the function. The next terminal down is the first parameter passed by reference, etc.

3 Enter the correct DLL function name in the Function Name field.

For example, for a DLL function defined as

```
long foo(double *x, double y, long *z);
```

you need three input terminals for x, y, and z and three output terminals, one for the return value and two for x and z. The Function Name field would contain foo. If the number of input and output terminals does not exactly match the number of parameters in the function, VEE generates an error.

If the DLL library has already been loaded and you enter the function name in the Function Name field, you can also use the Configure Pinout selection on the Call object menu to configure the terminals.

The Delete Library Object

If you have very large programs you may want to delete libraries after you use them. The Delete Library object deletes libraries from memory just as the Delete Lib selection on the Import Library object menu does.

Using DLL Functions in Formula Objects

You can also use DLL functions in formula objects. With formula objects, only the return value is used in the formula. The parameters passed by reference cannot be accessed. For example, the DLL function defined above is a formula:

```
4.5 + foo(a, b, c) * 10
```

where `a` is the top input terminal on the formula object, `b` is next, and `c` is last. The call to `foo` must have the correct number of parameters or VEE generates an error.

About Remote Functions

A Remote Function is a UserFunction that runs in another VEE process on a remote host computer. Remote Functions are a special case of UserFunction. See “About UserFunctions” on page 61 for general information that applies to UserFunctions.

Using Remote Functions

The Remote Function is called from the local VEE process over the LAN. Just as for UserFunctions and Compiled Functions, import a library of Remote Functions with the Import Library object.

When one or more Remote Functions have been imported, they are called by using the Call object or by including function names in expressions. You include Remote Function calls in your program just as you would UserFunctions. However, some differences and some networking technicalities are described in this section.

Create a library of Remote Functions just as you would a library of UserFunctions, but save it on the intended remote host computer. The intended remote host computer must also have VEE Pro or VEE Pro Run Time installed on it.

The library of Remote Functions is imported not into the local VEE process but in a special invocation of VEE called a "service" that runs on the remote host. The local VEE process is called the "client."

The client VEE process imports the Remote Function library using the Import Library object. When you select Remote Function for the Library Type field, some new fields appear as shown in Figure 7.

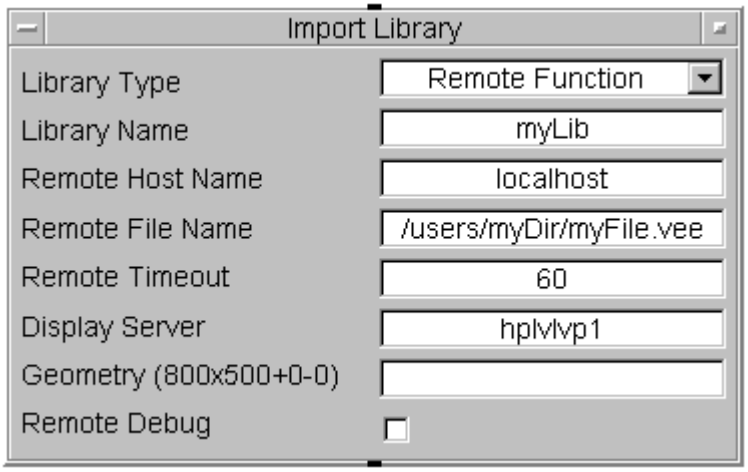


Figure 7 Import Library for Remote Functions

The Library Type and Library Name fields function the same as for UserFunctions and Compiled Functions. The other fields are as follows:

- Remote Host Name - The name of the host on which the "service" VEE process is to run (the "remote host"). This name can be the common or symbolic name of the host (for example `myhost`) or the IP address of the host in this field (for example `14.13.29.99`).
- Remote File Name - The name of the Remote Function library file. The Remote File Name is analogous to the File Name field for a UserFunction library. However, you must specify the *absolute* path to the file. Hence the path and file name can be rather long. You may want to have all users place remote function library files in a common place, such as:
  C:\USERS\REMFUNC.

NOTE

The remote VEE service invoked by the client is dependent on the Host Name specified in the Import Library object. If you have two Import Library objects using the same Host Name, only one service process is invoked. Even if two different Library Names and Remote File Names are used, each communicates with the same service. On the other hand, if each Import Library uses a different Host Name, two separate services are invoked.

- Remote Timeout - A timeout period in seconds for communication with the VEE service. If the VEE service has not returned the expected results of a Remote Function within this time period, an error occurs.
- Remote Debug - When this check box is selected, all UserFunctions within the library execute in debug mode (i.e., you will be able to perform debugging on them, such as setting breakpoints and doing line probes). This setting works with UserFunctions whether or not they have panel views.

When the Import Library object is executed (either by selecting Load Lib from the object menu or during normal program execution), a VEE server process is started on the remote host specified in the Host Name field. The client process and the server process are connected over the network and are able to communicate.

When a Call object in the client VEE calls a Remote Function, the arguments (the data input pins on the Call object) are sent over the network to the remote service, the Remote Function is executed, and the results are sent back to the Call object and output on its data output pins.

If your program deletes the library of Remote Functions with the Delete Library object, the Remote Functions associated with the library are removed. You can load multiple libraries in a VEE server process, then delete each one as needed without canceling the service connection. The VEE server exists while the VEE client process continues to run.

The service VEE process can exist on the same computer or "host" as the client or on another host as long as there is a network connection between them. The most common connection is between two hosts on a LAN. However, if a network path exists, the two hosts could be a continent apart.

The VEE service process has some attributes that are different from a normal VEE process:

1. The VEE service process executes only Remote Functions that are contained in the Remote Function library named by Import Library.
2. Remote Functions have views associated with them. When you call a remote function, you can have a VEE window appear on the remote host if the UserFunction displays a panel view.
3. Global variables (declared and undeclared) are not shared between the processes.
4. Remote Functions do not time-slice when called.
5. Parameters of type object cannot be passed to or from a Remote Function (includes ActiveX Automation objects or pointers to ActiveX controls).
6. The Execution Mode used by the service VEE process is that of the user's .veerc file, not that saved in the file that is imported.
7. Embedded .veeio file configurations in the file imported by the service VEE process are ignored. Only the global I/O configuration file is used.

You have to start the VEE Service Manager manually, as follows:

1. Go to the VEE installation directory
2. Execute *veesm.exe*
3. When the console window appears, you can minimize it to get it out of the way
4. To stop the VEE Service Manager process, open the console window and press `Ctrl+C`.

To automate the VEE Service Manager startup:

1. Create a shortcut to *veesm.exe*
2. Select **Start ⇒Programs**
3. Move the shortcut to the Startup folder.

www.agilent.com

Contact us

To obtain service, warranty or technical assistance, contact us at the following phone or fax numbers:

United States:
(tel) 800 829 4444 (fax) 800 829 4433
Canada:
(tel) 877 894 4414 (fax) 800 746 4866
China:
(tel) 800 810 0189 (fax) 800 820 2816
Europe:
(tel) 31 20 547 2111
Japan:
(tel) (81) 426 56 7832 (fax) (81) 426 56 7840
Korea:
(tel) (080) 769 0800 (fax) (080) 769 0900
Latin America:
(tel) (305) 269 7500
Taiwan:
(tel) 0800 047 866 (fax) 0800 286 331
Other Asia Pacific Countries:
(tel) (65) 6375 8100 (fax) (65) 6755 0042

Or visit Agilent worlwide web at:
www.agilent.com/find/assist

Product specifications and descriptions in this document are subject to change without notice. Always refer to Agilent Web site for the latest revision.

Printed in Malaysia
July 21, 2008

U2351-90701

Agilent Technologies