Hardware Abstraction

A hardware abstraction layer is a layer of programming that allows a system to interact with a hardware device at a general level rather than at a detailed level. Because the hardware is conceptually disconnected from the software, a developer can write software without critical consideration of the hardware's behavior. At some point, however, the hardware must be identified and connected to the software so that the appropriate drivers can be used for hardware operation. The configuration file (config.ini) is the file that creates the driver coupling to the hardware.

Config.ini File

The config.ini file is designed to provide the hardware coupling to the execution software at run-time. It is designed to contain specific required fields that are universally implemented with hardware, as well as any additional custom fields that may be unique to a specific device. The critical components in config.ini are the section header (Device Handle), Key Names, and Key Values. All categories are stored in the config.ini file as string values. **Note: Add information on critical keys: Class, Resource, Calibration.

Device Handle

The device handle field is used to create unique names for the installed devices. Unique names are required because there are commonly multiples of the same device installed in a test station. Creating a unique name allows for independent controls to be created for any or all duplicate devices. Device handles are contained within square brackets '[]'. Figure 1 shows a general setup for a power supply. The device handle in Figure 1 is 'Power Supply'. This is a generic name that should be avoided, but is useful for unit testing developed drivers.

Key Names

Key names are the designators for values stored for a device. These are designed to be human readable labels for the values that are stored in the config.ini file. Key names must be unique within a single section, but can be repeated in separate sections. Key names are followed by the equal '=' sign. Figure 1 shows the key names for [Power Supply] are Class, Calibration, Resource, and Channel.

Key Values

Key values are the critical data points associated with the key names. Key values can be identified by the values stored in quotation marks ‘ ”” ’ in the config.ini file.

Requisite Values

It is important to note that every device contains certain fields. These fields are required because they are definition fields that identify the critical details of the device. The required Keys are: Class, Calibration, and Resource.

Class

The class field is the most critical field in each section. This field informs the program which drivers are necessary to interact with the device. From Figure 1, the Class value is 6700C. This informs the program that when it creates a reference to [Power Supply], it must load the 6700C methods to control the power supply.

Calibration

The calibration field is required to simplify periodic maintenance on a complete test station. The configuration library reads the calibration date and determines whether or not the device requires calibration, and determines if the device is still within its calibration cycle. In Figure 1 the [Power Supply]’s Calibration due date is “NCR” which stands for “No Calibration Required”. The NCR value is useful for proving software functionality without necessarily tracking an instrument’s calibration cycle. This field will likely be updated when the test station is deployed.

Resource

The resource field is used to establish the communication to the device. Every device has a unique communication format that is defined by the device specific class. In Figure 1, the [Power Supply]’s communication resource is “TCPIP0::192.168.200.1::INSTR0”.

Additional Keys

Some devices have additional fields that must be defined prior to launch. This is where additional keys can be added to the configuration file. From Figure 1, the additional key Channel is configured to “1”.

Understanding the Whole System

Putting together the information outlined in 2.4.1:4 and pulled from Figure 1, we can identify the configuration for the [Power Supply] as: A N6700C Power Supply with no periodic calibration cycle, that uses TCP/IP communication and has the IP Address 192.168.200.1, and this class is only controlling channel 1.

Configuration Editor Tool

Each revision of the configuration packed project library includes a configuration editor executable. This tool allows a developer to modify the configuration file programmatically and mitigates the risk of mis-keying critical fields. The run time tool should be placed in a folder in proximity to the targeted config.ini file to be modified. This is usually done by creating a folder for the config.ini file in a root directory and a separate folder for the configuration PPL and editor in the same root. This is also the required folder hierarchy defined by the programming guidelines document.

When the configuration tool is launched, it will search for the closest config.ini file and load the configuration data into memory and launch the GUI shown in Figure 2. From this screen, the developer can view all unique instruments defined in the config.ini file. The developer can add instruments to the configuration file, edit existing configuration settings, or remove configured instruments from the file.

Adding Instruments

Adding a device to the config.ini file is a guided process when using the configuration editor tool.

Create Unique Device Name

Once the add device button has been pressed, the dialog box shown in Figure 3 pops up; prompting the developer for a unique device name. The tool verifies that the device name entered is unique and prevents duplicate entries.

Class Identifier

After a unique device name has been created, a browse utility is opened for the developer to locate the specific **Child** class associated with the device.

Calibration Data

The next prompt is to provide calibration information for the newly installed device. The control is a calendar browser and creates the calibration date information to the expected format for the configuration tool. No calibration required is an acceptable value for the configuration tool.

Communications Resource

The next prompt is looking for the device's communication resource. Several options are available in this prompt. First is a VISA resource browser, which identifies all connected instruments and their addresses. If the device is not found on the resource browser (such as TCP devices, PXI addressable devices, .NET devices, or similar), the operator has the opportunity to type in custom resource information as the child class expects it. Additionally, 'No Resource' is an acceptable option for administrative tools that do not use communication resources.

Additional Keys

After the critical keys have been created for the device, the key editor tool pops up (Figure 7). This tool allows the user to verify the configuration as defined as well as add, modify or remove keys. The Class key is protected and cannot be deleted, it can, however, be modified. If an additional key is required select 'Add'.

Class specific keys that are added do not have any controls with respect to spelling, so it is imperative that the developer be confident in their data entry at this point.

Recommendations

If any issues are identified while using this tool, please contact the author so the source code can be updated and new revisions can be deployed.