CC-USB based Data Acquisition System

Ron Fox
CC-USB based Data Acquisition System
by Ron Fox

Revision History
Revision 1.0 February 3, 2008 Revised by: RF
Original Release
Revision 2.0 January 18, 2010
Added information about additional modules and made it closer to what a class might need
Revision 2.1 March 24, 2010
Document support for CAEN C1205 and CAEN 257 modules.
List of Examples

3-1. Sample configuration and parameters definitions .................................................................9
3-1. AD811 configuration file example ..........................................................................................11
3-1. LRS2228 creation example ..................................................................................................17
3-1. The lrs2249 command ......................................................................................................18
3-1. The lrs2551 command ......................................................................................................20
3-1. Using the list command to construct pedestals .................................................................22
3-2. Sample ph7xxx commands ..................................................................................................23
3-1. Example of the stack command ..........................................................................................26
3-1. Making a correspondence between module channels and parameters ...............................27
3-10. A complete daqconfig.tcl file .........................................................................................28
4-1. Using the Tcl list command ................................................................................................31
4-2. Incorporating automatic parameter/spectrum generation into SpecTcl .................................31
Chapter 1. Introduction

This chapter is an orientation to the CC-USB based data acquisition system.

- The Section "CC-USB description" provides an overview of the CC-USB controller module
- "Supported Devices" provides information about which data taking devices are supported by this release and any support restrictions.
- "Script organization" is an overview to how configuration scripts cooperate to supply configuration for readout, and the NSCL histogramming program; SpecTcl.

1.1. CC-USB description

The CC-USB is a CAMAC controller that connects to a host system via a USB-2 interface. USB-2 is capable of transfer rates of 480Mb/sec, however the USB protocol has a very high transfer initiation latency. This makes USB-2 unsuitable for directly interacting with data taking devices on an event by event basis.

The CC-USB implements local intelligence via a capability described in its manual as a stack. A stack is essentially a list of CAMAC actions stored in the CC-USB. The execution of a stack is triggered by an external condition. Stacks run without host interaction, filling buffers of data which can then be transferred to the host computer. Stacks are the CC-USB’s way to work around the high latencies of USB-2.

The CC-USB supports two stacks, and event stack and a periodic stack. The event stack is executed in response to an external trigger. The external trigger can be a CAMAC LAM (look at me) or, more commonly, a NIM logic true pulse on IN 1. The periodic stack can be execute either time or event periodically (by event periodic I mean that after some number of event triggers have been accepted, the periodic stack is executed). The periodic stack is intended to be used to read scalers. Some setups may use scalers as counters to monitor detector rates, or to determine acquisition efficiency (live-time).

Stacks are rather hard to construct by hand. The CC-USB system automates stack construction through configuration files. Configuration files allow you to define your setup in terms of the modules you want to use, and the order in which they should be read in response to the stack’s trigger. Configuration files not only remove the error prone nature of stack construction but allow you to think of your data acquisition problem in terms of the devices you use and how you will use them.

The section Script organization, later in this chapter describes how configuration files, or scripts, are used by the acquisition and analysis software. Chapter 3 - Configuration Files describes configuration files in detail.
1.2. Supported Devices

One consequence of using high level configuration scripts to describe your experiment is that the scripts can only support a pre-determined set of CAMAC modules. Each module requires driver software that understands the readout requirements of its module and unpacking software that understands the format of the data that is provided by that module.

The following devices are supported:

1. Ortec AD811 peak sensing ADC.
2. Phillips 7164/7164H. This is a 16 channel peak sensing adc.
3. Phillips 7166/7166H. This is a 16 Channel charge integrating ADC (QDC).
4. Phillips 7167/7167H. This is a 16 channel individually gated charge integrating ADC (QDC).
5. Phillips 7186/7186/H. This is a 16 channel time digitizer (TDC).
6. LeCroy Research Systems (LRS) 2228. This module is a time interval digitizer. It can be used in either common start or common stop mode.
7. LeCroy Research Systems (LRS) 2249W ADC. This module is a charge integrating ADC.
8. LeCroy Research Systems (LRS) 2551 scaler module. This module is normally placed in a scaler stack.
9. CAEN C1205 QDC
10. CAEN 257 scaler module. This module is normally placed in a scaler stack.

The architecture of the system is such that additional module types can be supported at a later time if desired.

1.3. Script organization

A configuration file is used to drive both the generation of CC-USB stacks and the unpacking of event data. Configuration files are written in an extension of the Tcl\(^1\) (Tool Command Language) scripting language. As such you have a fully featured programming language at your disposal when you are building your configuration.

Configuration files all are located in the config directory of your account (~/.config to be precise). The file you will usually have to edit is ~/.config/daqconfig.tcl or daqconfig for short.

Each time you start data taking, the readout software will interpret daqconfig.tcl, initialize the CAMAC modules you are using, create and download the stacks you describe. You can therefore update a configuration file and the update will be effective the next time you start a run.
SpecTcl histogrammer will interpret the daqconfig.tcl script, create parameter definitions and spectra for each of the digitizer channels you define in your configuration. SpecTcl will also configure its event decoder to unpack your raw events into the parameters you have described in daqconfig.tcl. Since SpecTcl will only interpret daqconfig.tcl when you start it, you will need to restart SpecTcl when:

- You define new parameters in existing digitizers and want to see the spectra from these parameters.
- You alter the set of modules or the order of modules present in the event stack.

Notes

1. For a Tcl tutorial that goes well beyond what you will need, see http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html (http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html)
Chapter 2. Getting Started

This chapter provides a quickstart for people who just want to get off the ground and take some data. When you have finished reading this chapter you should know:

- The components of the NSCL Data Acquisition system (NSCLDAQ) you will be using.
- How to configure your computer account for data taking.
- How to edit `daqconfig.tcl` to prepare to take data.
- How to modify the sample scaler configuration file.
- How to set up a trigger for the CC-USB event list.
- How to start up each component of the data acquisition system.

2.1. Introducing NSCLDAQ

The NSCL Data Acquisition system (NSCLDAQ) is a distributed data acquisition system. A single system provides data from a data source to an arbitrary set of client systems. We will use this system. For the purposes of this discussion, all client software will run in the data source system. Multi-computer data taking systems are beyond the scope of this document.

The NSCLDAQ is a flexible system that consists of many components. When you use the CC-USB for data taking, every effort has been made to provide near turnkey operation. If you are interested in reading more about NSCLDAQ, documentation is online at http://docs.nscl.msu.edu/daq.

When you use the CC-USB readout system you will only need to know about the Readout, SpecTcl and scaler display components of the system. If you record data to disk, the event logging component will run behind the scenes automatically.

2.2. Configuring your account for data taking

In order to minimize what you need to know to get started, a script has been provided that configures your account for data-taking. This script:

- Creates the directory structure expected by the CC-USB data acquisition system.
- Copies some startup scripts into your account’s `bin` subdirectory.
- Creates the `config` directory and copies sample configuration files into it, as well as support scripts for SpecTcl’s configuration file decode.
- Creates the `spectcl` directory and sets it up as the default directory in which SpecTcl will be run by copying initialization scripts that ensure SpecTcl will process `daqconfig.tcl` when it is started.
Chapter 2. Getting Started

- Sets up ssh host/identity keys so that you can ssh localhost without providing additional login credentials. This is used by the GUI that runs the readout program (remember NSCLDAQ is a distributed system and the GUI does not assume the Readout program will run on the same system it runs on).

To set up your account for data taking, the first time you use it, login, bring up a terminal window and type:

```
/usr/opt/ccusbsetup/daqsetup
```

When you are prompted to do so accept the host key for localhost, then logout of the 'remote' session the script set up to exchange those keys.

### 2.3. Editing daqconfig.tcl

The core configuration file is `daqconfig.tcl`. A sample `daqconfig.tcl` file was copied to your `config` directory when you set up the account.

The sample configuration file uses one of each type of module and sets up an event as well as a scaler stack. You can follow its example to create your own configuration file. If you get stuck, have a look at the reference material in the chapter 'Configuration Files'. In general:

- Ensure the modules `-slot` number matches the CAMAC crate slot in which you’ve inserted the module.
- Ensure the `-id` value is unique across all modules that are in the events stack.
- Make sure that all modules you want read out in response to the event trigger are in the `[list ...]` that follows the `-modules switch on the stack config events line of the file. Make sure that only those modules are listed that you want to read out as well. If you define a module, but don’t list it in either stack it will be ignored. List modules in the order in which you want them read.
- Be sure that if you use any scaler modules you have listed them in the `[list ]` part of the `stack config scaler` command. List them in the order you want them read.
- If you are not using scaler modules, remove the two lines that describe the scaler stack `stack create scaler and stack config scaler....`

If you do use the scaler stack you can adjust the readout period by setting it in seconds after the `-period switch.

- If you add modules, you will want to add to the `for` loops below the module and stack definitions. These sections define meta-data that is used by SpecTcl to set up parameters and spectra.
2.4. Editing the scaler configuration file

The scaler configuration file is only needed if you are going to read and display scaler data. If you are not going to do this, skip this section.

When you set up the account for data taking the setup script created the file `config/scalers.def`. This file is a sample scaler definition file. Look carefully at the comments at the top of the file. The comments describe how to edit this file to meet your needs.

2.5. Setting up the CC-USB trigger

Part, but not all of the electronics setup includes providing a trigger to the CC-USB. The trigger defines when the list of modules described in your stack for event readout will be read.

To trigger CC-USB readout, send a NIM true pulse into the CC-USB’s I1 input. In addition, you will typically need to provide gates to the CAMAC modules to tell them to start converting. Use the `-delay` switch on the event stack command to allow all of your CAMAC modules to finish converting before starting to execute the event stack. The value for this is the number of microseconds the CC-USB will delay between the I1 pulse executing the event readout list. If you do not set the `-delay` value large enough, you may start reading the CAMAC modules before they have finished converting.

2.6. Starting up NSCLDAQ components

The account setup script installs several scripts in the account’s `~/bin` directory that know how to start up the components of NSCLDAQ you needs. Specifically.

`startReadout`

Starts the readout GUI and the CC-USB readout program. The GUI has buttons that allow you to start/stop runs, record data or not. The Readout program will read the file `~/config/daqconfig.tcl` at the beginning of each run to determine how to setup the CC-USB for data taking.

`startSpecTcl`

Starts the NSCL SpecTcl histogramming program. Please note that you must still attach SpecTcl to the online NSCLDAQ system via the Data Source menu.

SpecTcl will process the `~/config/daqconfig.tcl` file only once, as it start up. If you make changes to that file that might change the event structure, you must restart SpecTcl.
Chapter 2. Getting Started

startScaler

Starts the NSCL Scaler display program. This command processes the ~/config/scalers.def file. This file provides names for each scaler channel, defines display pages and allows you to define which scalers, and scaler ratios appear on which pages.

The script connects the scaler program to the NSCLDAQ online system automatically.
Chapter 3. Configuration Files.

This chapter describes the configuration files that drive Readout and SpecTcl. The end of this chapter contains reference material that describes the extensions to Tcl that have been incorporated into the Readout configuration script engine to support building CC-USB stacks directly from the configuration file ~/config/daqconfig.tcl.

3.1. daqconfig.tcl The readout config file

The ~/config/daqconfig.tcl file is the basis for configuring the Reaout (directly), and SpecTcl (indirectly). This file is a script for a Tcl interpreter that has been extended with additional commands that support describing the hardware to be readout and how to aggregate that hardware into stacks.

Extensions to Tcl for configuration are a group of commands. The command keyword for each command indicates the type of device or item it manipulates. For example the ph7xxx command creates, configures or gets the configuration of the various supported Phillips digitizer modules.

The extensions to Tcl are what Tcl calls command ensembles. A command ensemble is a command with subcommands. Each extension has three subcommands:

create
  Creates an item of the type associated with a command

config
  Configures an item of the type associated with a command

getc
  Returns configuration information about an item of the type associated with a command.

The following are the commands that have been added to the Tcl interpreter. See the command reference at the end of this chapter for documentation of each command.

ad811
  Manages the Ortec AD811 8 channel peak sensing ADC. This command could probably also work with an ORTEC TD811 8 channel TDC as the two modules are software compatible.

lrs2228
  Manages the LeCroy Research System LRS 2228 TDC.
Chapter 3. Configuration Files.

lrs2249

Manages the LeCroy Research Systems LRS 2249 QDC. This module comes in subtypes as well indicated by a letter after the 2249 model number. This command works for all types of LRS 2249. Testing was done with the LRS 2249W.

lrs2551

Manages the LeCroy Research System LRS 2251 scaler module. This module is normally read in the scaler stack, however in special application (not supported by the automatic SpecTcl unpacking software), this module could be read out by the event stack.

ph7xxx

Manages Phillips 16 channel digitizers. These include the 7164 (ADC), 7166, 7167 (QDCs), and 7186 (TDC) modules.

c1205

Manages CAEN 1205 16 channel QDC modules.

c256

Manages CAEN 257 scaler modules.

stack

Used to construct readout and scaler stacks.

3.2. Configuring SpecTcl

Using the helper file ~/config/spectclconfig.tcl, SpecTcl’s startup scripts are able to process enough of the ~/config/daqconfig.tcl to be able to create a correspondence between module channel numbers and parameter names. The ~/config/spectclconfig.tcl script provides Tcl proc extract the order in which the modules have been added to the event readout stack. This file is installed when the account was set up as a data acquisition account.

When ~/config/spectclconfig.tcl processes ~/config/daqconfig.tcl it expects that file top have set the global variable parameters. This variable is an array that is indexed by module name. Each element contains a list of names of the channels for each digitizer you are using. Each of these lists can terminate early if the last channels in an module are not used. Intermediate unused channels should be named ""; an empty string. ¹

Example 3-1. Sample configuration and parameters definnitions

ph7xxx create adc1 -slot 6 -id 1
ph7xxx create adc2 -slot 7 -id 2

stack create events
stack config events -type event -modules [list adc1 adc2] -delay 10 -lamtimeout 8

¹
In the example above, two modules named `adc1` and `adc2` are created. At the bottom of the file we can see that for each `adc` named `adc n`, we have defined the names of all 16 channels to be: `ph n.00` through `ph n.16`. The `tcl list` command creates a list of items. The `[...]` in Tcl substitutes the results of the command enclosed in the square brackets in place on the current command.

The SpecTcl initialization file `spectclconfig.tcl` processes this configuration to produce SpecTcl parameters and 1-d spectra for each of these parameters.

### 3.3. Script Reference

This section contains reference information about the commands that have been added to Tcl to support configuring the readout. In addition a reference to the set of meaningful `daqconfig.tcl` script variables is supplied.

Please note that in some cases you must either specify specific options for SpecTcl to be able to decode the data or allow those options to default. When this is the case, the correct settings are indicated.

**ad811**

**Name**

`ad811` — Support the Ortec AD811 ADC

**Synopsis**

`ad811 create module-name ?option...?`
ad811 config module-name ?option...?

ad811 cget module-name

DESCRIPTION

This command ensemble provides support for the ORTEC AD811 8 channel peak sensing ADC. This module requires about 80 microseconds to convert. All channels convert unconditionally.

module-name is either the name to be given to an ADC in the create subcommand or the name given to a previously created module in the config or cget commands.

Both create and config take configuration options. Each option is a keyword value pair where the keyword defines what the option is and the value is a value associated with that option. See OPTIONS below for more information about the options supported by ad811.

The cget option returns the configuration options for the named module as a Tcl list. Each element of the list is a two element sublist. The first element of each sublist is the configuration parameter name, and the second the value associated with that name.

OPTIONS

The ad811 configuration options are:

-id vsn
    Each module has an identifier which, for historical reasons is called a virtual slot number, or vsn. To maximize the decode error detection the virtual slot numbers for each module should be unique. This option allows you to set the virtual slot number for the module.

-slot slotnum
    The value of this parameter tells the readout software which CAMAC crate slot houses the module. Recall that CAMAC is slot addressable. Slots number from 1 on the left, with an increasing slot number towards the right. The right two slots are reserved for the CAMAC Crate controller (in our case a CC-USB).

EXAMPLE

The example below creates and configures an AD811 that is in slot 2 of the CAMAC crate and, when read will have a virtual slot number of 2:
Example 3-1. AD811 configuration file example

ad811 create adc -slot 2 -id 2

Chapter 3. Configuration Files.

c1205

Name
c1205 — Manage CAEN C1205 QDC modules.

Synopsis
c1205 create module-name ?option...?

c1205 config module-name ?option...?

c1205 config module-name

DESCRIPTION

This command ensemble allows you to create and configure CAEN model C1205 QDC modules. The C1205 is a sixteen channel charge integrating ADC.

The create command allows you to create a new module and assign a module-name to it so that it can be referred to in subsequent configuration commands. You may optionally include configuration switch value pairs on the create command line.

The config subcommand allows you to set the configuration parameters for an existing module. module-name is the name assigned to that module when it was created.

The cget command returns as its value the module configuration as a Tcl list where each list element is a sublist containing the name and value of a configuration parameter.

Configuration options are name/value pairs. The name of a configuration option is sometimes called a switch because it resembles the command line switches of command shell commands. For a full description of these options see OPTIONS below.
OPTIONS

The CAEN C1205 supports the following configuration option switches. The switches and values can be supplied to both the `create` and the `config` subcommands.

- **-slot n**
  Specifies the slot in which the module is installed to `n`. If this is not set correctly, the Readout program will not be able to communicate with the module and will emit a warning, if the slot `n` is empty. This parameter must be provided.

- **-id n**
  Sets the id of the module to `n`. The module id is placed as a marker word prior to the data from the module. It is used by the unpacking software to unambiguously identify the data from each module. For the C1205, the id value is also programmed into the lower 8 bits of the control status register and will therefore also appear in the bottom 8 bits of the header word read from the module.

  While the `-id` value defaults to 0, and need not be unique, your data is most reliably decoded if each module in your system has a unique id value.

- **-usepedestals true/false**
  The value of this configuration parameter determines how bit 12 of the control status register is initialized. If `true` (default value), the bit is set to one which enables pedestal subtraction from the input signal.

- **-hires true/false**
  The value of this configuration parameter determines how bit 16 of the module control status register is initialized. If `true` (default value) the bit is set to zero enabling high resolution mode. If set `false` low resolution mode is selected. High resolution mode provides 12 bits of data resolution while low resolution mode provides only 10 bits of data resolution. High resolution comes at a cost of an additional 1.5 microseconds of conversion time (5.5 microseconds compared with 4 microseconds for low resolution).

- **-thresholds threshold-list**
  Provides the channel threshold registers. This is a Tcl list of 16 integers in the range 0-4095. Data is read from a module only if the data value for the module’s low range is greater than the corresponding element of the list. The thresholds default to a list of 16 zeroes.

- **-lopedestals pedestal-list**
  Provides a list of 16 integers (0-4095) which are subtracted from the data values to produce the low range value.

- **-midpedestals pedestal-list**
  Same as `-lopedestals` but the pedestals are applied to the mid level conversion.
Chapter 3. Configuration Files.

-hipedestal pedestal-list
  Same as -lopedestals but the pedestals are applied to the low level conversion.

-rangemode mode
  Sets the module range mode. The value for mode must be one of the following:

  all
  Conversions from all ranges appear in each event. In this mode, some of the conversions will likely have negative values.

  auto
  In this mode (default) only the range the conversion falls in will appear in the data from the module.

  sparse
  Same as auto, but only data from channels are above the channel thresholds set via the -thresholds option appear in the data from the module.

Manage CAEN C257 scaler module.

Name
c257 — Manages the C257 scaler module

Synopsis
c257 create name ?options...?
c257 config name ?options...?
c257 cget name

DESCRIPTION
This command ensemble allows you to create and configure CAEN model C257 scaler modules. The C257 is a 16 channel scaler module. Support is intended to be used with the scaler stack. The C257 is capable of cascading channels to produce very wide counters, however this mode is not supported by this
release of the software limiting the width of each scaler to 24 bits. Thus, to use the module with this software requires that all of the module jumpers be set to the SINGLE position rather than to CASCADE.

The `create` command allows you to create a new module and assign a `module-name` to it so that it can be referred to in subsequent configuration commands. You may optionally include configuration switch value pairs on the `create` command line.

The `config` subcommand allows you to set the configuration parameters for an existing module. `module-name` is the name assigned to that module when it was created.

The `cget` command returns as its value the module configuration as a Tcl list where each list element is a sublist containing the name and value of a configuration parameter.

Configuration options are name/value pairs. The name of a configuration option is sometimes called a switch because it resembles the command line switches of command shell commands. For a full description of these options see OPTIONS below.

**OPTIONS**

The module is configured via the following set of options.

- `-slot slotnumber`
  Specifies `slotnumber` as the CAMAC crate slot in which the module is inserted. This must be correct or else the software will not be able to communicate with the module.

- `-id module-id`
  Specifies a module id for the unit. If the `-insertid` configuration parameter is true this id is inserted prior to the scaler data readout by the stack. This option, and `-insertid` should not be used in the scaler stack and is intended as latent support for using the scaler in an event stack (e.g. for supplying an event timestamp).

- `-insertid true/false`
  If this parameter is true the value specified by the `-id` option will be inserted in the data buffer prior to the data from this module. This should be false (the default) if the module is used in the scaler stack.

  `-id` and `-insertid` are intended as latent support for using the scaler module in the event stack (e.g. to supply an event timestamp).
Chapter 3. Configuration Files.

-`readinhibit true | false`

If this parameter is `true` (the default), the CAMAC crate is held inhibited while the scaler module is being read, and the module is read with an F2 Q-scan. If `false` the module is read via 16 F0’s directed at specific subaddresses followed by an F9 to reset the counters.

**Manage LRS2228 LeCroy TDC**

**Name**

*lrs2228* — Manages the LRS2228 TDC

**Synopsis**

*lrs2228 create module-name ?option...?*

*lrs2228 config module-name ?option...?*

*lrs2228 cget module-name*

**DESCRIPTION**

This command ensemble allows you to create and configure LeCroy model LRS 2228 TDC modules. The LRS 2228 is an eight channel Time digitizer.

The *create* command allows you to create a new module and assign a `module-name` to it so that it can be referred to in subsequent configuration commands. You may optionally include configuration switch value pairs on the *create* command line.

The *config* subcommand allows you to set the configuration parameters for an existing module. `module-name` is the name assigned to that module when it was created.

The *cget* command returns as its value the module configuration as a Tcl list where each list element is a sublist containing the name and value of a configuration parameter.
Configuration options are name/value pairs. The name of a configuration option is sometimes called a *switch* because it resembles the command line switches of command shell commands. For a full description of these options see OPTIONS below.

**OPTIONS**

The following options are recognized by the `lrs2228 create` and `lrs2228 config` commands.

- `id vsn`
  Each module should be assigned a unique virtual number. If not assigned this will be 0. The data packet returned by the module will be prefixed by the ID. This provides error checking for the code that will be unpacking the raw data.

- `slot camac-slot`
  Provides the number of the slot in which the CAMAC module has been installed (`camac_slot`). This must match the physical slot the module has been stuffed in in order to ensure the proper module is initialized and reads.

**EXAMPLES**

The following is an example of the `create` command for the LRS2228:

**Example 3-1. LRS2228 creation example**

```
lrs2228 create tdc -slot 6 -id 4
```

---

**Irs2249**

**Name**

`Irs2249` — Manage LeCroy 2249 QDC modules

**Synopsis**

```
lrs2249 create module-name ?option...?
```

```
lrs2249 config module-name ?option...?
```
The \texttt{lrs2249} command ensemble allows you to read out LeCroy model LRS 2249 modules. The LRS 2249 is a charge integrating ADC (QDC).

The \texttt{create} subcommand creates a new module assigning it the \texttt{module-name} you supply. The \texttt{module-name} can be used to refer to the module in future commands. You may optionally provide configuration information when creating the module.

The \texttt{config} subcommand configures an existing module by providing configuration options for an existing \texttt{module-name} (created via the \texttt{create} subcommand). Configuration options are name/value pairs that are also sometimes called switches because of their resemblance to shell command switches. For a list of the valid configuration options for the LRS 2249 module, see the OPTIONS section below.

The \texttt{cget} subcommand returns the configuration parameters for \texttt{module-name} as a Tcl list of items. Each item is itself a two element sublist containing in order the name and value of a configuration option.

\textbf{OPTIONS}

The \texttt{lrs2249} command has the following configuration options:

\begin{itemize}
  \item \texttt{-id vsn} \quad Sets the module virtual slot number to \texttt{vsn}. The virtual slot number prefixes the data packet read from this module and is used by SpecTcl as a check on the validity the event. Each module should have a unique \texttt{vsn}. If the \texttt{-id} option does not set the virtual slot it defaults to 0.
  \item \texttt{-slot slot} \quad Provides the module \texttt{slot}. This parameter must match the number of the slot in the CAMAC crate in which the module is installed.
\end{itemize}

\textbf{EXAMPLE}

The command below creates and configures an LRS 2249 QDC module:

\textbf{Example 3-1. The \texttt{lrs2249} command}

\begin{verbatim}
lrs2249 create qdc -slot 5 -id 3
\end{verbatim}
**lrs2551**

**Name**

`lrs2551` — Manage LRS 2551 modules

**Synopsis**

`lrs2551 create module-name ?options...?`

`lrs2551 config module-name ?options...?`

`lrs2551 cget module-name`

**DESCRIPTION**

The `lrs2551` command allows you to create and configure LeCroy LRS 2551 scaler modules. The LRS 2551 is a 12 channel scaler. While it is normally read in the periodic scaler stack, it is also possible to read it in the event stack when properly configured.

The `create` subcommand creates a new module assigning it the name `module-name`. `module-name` should be used to refer to the module in future commands. Optional configuration data can also be appended to the `create` subcommand.

The `config` subcommand configures an existing `module-name`. The configuration items are name value pairs. See OPTIONS below for more information about the options supported by the `lrs2551` command.

The `cget` subcommand returns the current module configuration. The configuration is returned as a Tcl list. Each element of the list is a two element sublist that contains the configuration name and value in that order.

**OPTIONS**

The `lrs2551` command supports the following configuration options:
-cumulative on|off

If -cumulative is false (default), the scaler module is cleared after each readout providing the counts between each readout as the value of each channel. This should be false when used with the NSCLDAQ scaler display program in the scaler stack.

-id vsn

Supplies the virtual slot number; vsn for the module. This only needs to be supplied if the module is going to be read in the event stack. Note that currently unpacking scaler data from the event stack is not supported.

-insertid true|false

If true (the default is false), the virtual slot number set by the -id configuration value is inserted in the data block from the scaler. If false it is not. If used with the NSCLDAQ Scaler display in the scaler stack, this should be false.

-slot slot

Sets the slot configuration. When installed in the CAMAC crate, the module should be inserted in the slot specified by this option.

EXAMPLES

This example creates an LRS 2551 scaler:

Example 3-1. The lrs2551 command

lrs2551 create counters -slot 4

ph7xxx

Name
ph7xxx — Define Phillips ADC/TDC/QDC modules

Synopsis

ph7xxx create name [?option value ...?]

ph7xxx config name option value...
Chapter 3. Configuration Files.

**ph7xxx cget name**

### DESCRIPTION

Creates, configures and interrogates the configuration of Philips CAMAC digitizers.

The **create** command creates a new digitizer *name* configuration. The *name* parameter must be unique. The optional [option value] pairs that follow specify the configuration for that module. This configuration can be supplemented or overridden by subsequent **config** operations on the module.

The **config** subcommand configures the existing module *name*. The remainder of the command line are option value pairs described in the section **OPTIONS** below. The module *name* must have already been created via the **create** subcommand. Note that configuration options are processed from left to right, if a configuration option appears more than once, the last instance takes effect.

The **cget** subcommand returns the configuration of the module. This can be used in more advanced scripts to analyze the configuration of the system. The configuration of a module *name* is returned as a properly formatted Tcl list. Each element of the list is itself a pair (a two element sublist). The first element of each pair is the configuration parameter name, the second element the value, which may itself be a list (e.g. for the **-llt** option).

### OPTIONS

- **-slot slotnumber**

  This option configures the slot in which the module will be installed. All modules must be in unique slots, although that is not verified by the configuration manager. The slot must be an integer number from one through 23 (slot 24/25 holds the CAMAC crate controller).

  The default value for this parameter is illegal (0), so you must specify this parameter.

- **-id vsn**

  Each module has an identifier which, for historical reasons is called a *virtual slot number*, or vsn. To maximize the decode error detection the virtual slot numbers for each module should be unique. This option allows you to set the virtual slot number for the module.

  If the **-id** is not supplied, it defaults to zero.
Chapter 3. Configuration Files.

-sparse bool
If the value of this configuration is true, the module is read in sparse readout mode (A Q-Stop of F4@A0). SpecTcl expects this to be true and that is also the default value.

The values true, yes, 1, on and enabled are all recognized as true values while the values false, no, 0, off, and disabled are all recognized as false values.

-readhits bool
If the value of this configuration is true, the module’s hit register is read prior to the channels (F6@A1). SpecTcl requires this parameter to be true, and that’s the default value.

SpecTcl will use this value to determine the number of digitizer data words that follow for this module.

The values true, yes, 1, on and enabled are all recognized as true values while the values false, no, 0, off, and disabled are all recognized as false values.

-pedestals int[16]
Supplies the pedestals for each channel. This must be a 16 element list of integers. These values are initialized to zero. Note that the pedestals are ignored unless -usepedestals is true.

The best way to construct this list of pedestals is to use the Tcl list command for example:

Example 3-1. Using the list command to construct pedestals

```plaintext
ph7xxx config someadc -pedestals [list 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20]
```

Note that pedestals can be specified as decimal values (as shown in the example above), hexadecimal value by preceding a number with the text 0x or octal by preceding the number with a leading 0.

-llt int[16]
Supplies the low level thresholds for each channel. This must be a 16 element list of integers. These values are initialized to zero. Note that the low level thresholds are ignored unless -usellt is true.

See the description of -pedestals above for more information about how to construct this list.
Chapter 3. Configuration Files.

-hlt int[16]

Supplies the high level thresholds for each channel. This must be a 16 element list of integers. These values are initialized to full scale (4095). The high level thresholds are ignored unless the parameter -usehlt is true.

See the description of -pedestals above for more information about how to construct this list.

-usellt boolean

Enables or disables the low level threshold (sets or clears the LT Enable bit in the control register). This is useful if -llt has been used to program low level thresholds and -sparse has been set to true.

The values true, yes, 1, on and enabled are all recognized as true values while the values false, no, 0, off, and disabled are all recognized as false values.

-usehlt bool

Enables or disables the high level threshold (sets or clears the UT Enable bit in the module control register). This is useful if -hlt has been used to set the high level thresholds and -sparse has been set to true.

The values true, yes, 1, on and enabled are all recognized as true values while the values false, no, 0, off, and disabled are all recognized as false values.

-usepedestals bool

Enables or disables per channel pedestal subtraction. This is most useful when -pedestals has been used to program a set of pedestals into the module.

The values true, yes, 1, on and enabled are all recognized as true values while the values false, no, 0, off, and disabled are all recognized as false values.

EXAMPLES

The example below creates a Phillips module named adc1 and programs all of the configuration options. Note that in many cases you won’t need to do this (e.g. for ADC’s the high level thresholds usually can be left up at 4095
Example 3-2. Sample ph7xxx commands

```bash
ph7xxx create adc1 -slot 5
ph7xxx config adc1 -sparse enabled -readhits true -usellt true -usehlt false
ph7xxx config adc1 -usepedestals enabled
ph7xxx config adc1 -pedestals [list 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ]
ph7xxx config adc1 -llt [list 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10]
ph7xxx config adc1 -hlt [list 3000 3000 3000 3000 3000 3000 3000 \ 
3000 3000 3000 3000 3000 3000 3000 3000]
```

stack

**Name**

stack — Create and configure CC-USB stacks.

**Synopsis**

```
stack create name
stack config name option value...
stack cget name
```

**DESCRIPTION**

Creates and configures CC-USB stacks. Stacks are lists of CAMAC operations that are loaded into the CC-USB for autonomous execution. There are two types of stacks; an event stack that is executed when a front panel trigger or a specified LAM is detected, and a scaler stack that can be periodically triggered.

Stacks contain lists of modules. The modules themselves determine the set of readout instructions that make up each stack.

**Note:** In this version of the U. Mass. Lowell software, no CAMAC scaler modules are supported, so the scaler stack is not very useful.
The create subcommand creates a new stack. The name of the new stack is \textit{name}. This \textit{name} will be used to refer to this stack in future configuration commands.

The config subcommand configures the stack \textit{name}. The configuration is specified as a set of option value pairs. The options are described in the section OPTIONS below.

The cget subcommand returns the configuration of the stack \textit{name} as its command value. This subcommand is intended for advanced applications that may analyze the configuration of the stack. The configuration is returned as a properly formatted Tcl list. Each element of the list is itself a two element sublist (pair). The first element of each pair is the name of a configuration parameter the second, that configuration parameter's value. Note that the value may itself be a list.

\section*{OPTIONS}

Options are used to configure the stack. Each option has associated with it a keyword that selects it and a value. The options supported by the \texttt{stack} command are subdivided into three groups:

1. Options used for both event and scaler stacks.
2. Options that are only used for event stacks and are ignored for scaler stacks
3. Options that are only used for scaler stacks, and are ignored for event stacks.

\subsection*{Options for all stack types.}

\texttt{-type \texttt{stack-type}}

Defines the type of stack that is being used. This can be one of the keywords \texttt{event} for event stacks or \texttt{scaler} for scaler stacks. The CC-USB only supports one scaler and one event stack. The current version of the configuration engine allows you to create more than one of each stack. In that case, it is not well defined which stacks are actually loaded.

The default \texttt{-type} is \texttt{event}.

\texttt{-modules \texttt{module-list}}

This option defines the set of modules that are added to the stack. \texttt{module-list} is a properly formatted Tcl list of names of modules (ph7xxx's in this implementation) that will be read by this stack. See the EXAMPLES section below for more information.

\subsection*{Event stack options.}

\texttt{-lams \texttt{mask}}

Specifies the mask of LAMs that can trigger the list. I suggest leaving this value at zero, and deriving a trigger signal for the NIM IN1 input as the readout trigger.
Chapter 3. Configuration Files.

-lnamtimeout microseconds

The ph7xxx command delays the read of the module until it is signalling a LAM. microseconds specifies the total number of microseconds the stack should wait for this LAM. The value should be on the order of the conversion time of the module. Defaults to zero.

-delay microseconds

Delays the execution of the stack for microseconds microseconds after the trigger condition is detected. This can be used to ensure the digitization hardware has sufficient time to complete its digitization prior to stack execution. Defaults to zero.

Options only available for scalers stacks. The options below are only legal for stacks of -type scaler

-period seconds

Number of seconds between execution of the scaler stack. This defaults to 2.

EXAMPLES

The example below extends the example in the ph7xxx command reference reading the defined modules in an event stack:

Example 3-1. Example of the stack command.

```
ph7xxx create adc1 -slot 6
ph7xxx create adc2 -slot 7

stack create events

stack config events -type event -modules [list adc1 adc2]
stack config events -delay 10 -lamtimeout 8
```

1 These two lines define the modules that we will be using for the experimental setup.
2 Creates a new stack called events In subsequent lines I’m going to configure this stack to handle the event trigger.
3 The stack is configured to be the event stack, and the two modules we created adc1 and adc2 are added to the stack in that order. When the stack is loaded, it will contain the instructions to read out these two modules.
4 Configures the stack so that the CC-USB will start stack execution 10 microseconds after the IN1 trigger. Stack instructions that wait for a lam on their target module will wait at most 8 microseconds. Since the -lams option is left to default to zero, CAMAC lams will not be able to trigger the list.
Variables

Name

Variables — Variables to set in the script

Synopsis

set parameters(module-name) channel-list

DESCRIPTION

When SpecTcl sets up its event decode software, it needs to know how to unpack specific channels of a CAMAC module into meaningful parameters. SpecTcl parameters have names, therefore, this means assigning SpecTcl parameter names to specific input channels of each module. So that ~/config/spectclconfig.tcl can issue SpecTcl commands that create this channel to parameter name correspondence.

~/config/spectclconfig.tcl does this by using a Tcl array. In Tcl, array indices are text strings, not numbers. It makes sense, therefore, to create an array, named parameters whose indices are the names of the modules that are being read out, and whose contents are a list of the channel names in that module.

Given this arrangement, SpecTcl can examine the -modules configuration of the event stack to determine the order in which modules will appear in the event, and then use the parameters array contents to know how to map each digitizer channel in the event to a SpecTcl parameter.

EXAMPLES

This example shows how to set up the parameters array to define a correspondence between the channels of the modules defined in the ph7xxx reference page example, and SpecTcl parameters.

Note that SpecTcl parameter names can be essentially any string. Periods in the string are used to define a parameter hierarchy that is reflected in the SpecTcl GUI.

Example 3-1. Making a correspondence between module channels and parameters

ph7xxx create adc1 -slot 6
ph7xxx create adc2 -slot 7
...

...
Chapter 3. Configuration Files.

3.4. A complete `daqconfig.tcl` file

In this section I will present a complete sample `~/config/daqconfig.tcl` configuration file.

**Example 3-10. A complete `daqconfig.tcl` file**

```tcl
ph7xxx create adc1 -slot 6
ph7xxx create adc2 -slot 7

stack create events
stack config events -type event -modules [list adc1 adc2] -delay 10 -lamtimeout 8

set parameters(adc1) [list ph1.00 ph1.01 ph1.02 ph1.03 ph1.04 ph1.05 ph1.06 ph1.07 ph1.08 ph1.09 ph1.10 ph1.11 ph1.12 ph1.13 ph1.14 ph1.15]

set parameters(adc2) [list ph2.00 ph2.01 ph2.02 ph2.03 ph2.04 ph2.05 ph2.06 ph2.07 ph2.08 ph2.09 ph2.10 ph2.11 ph2.12 ph2.13 ph2.14 ph2.15]
```

1. Defines the two adc modules from the original example. I’m going to assume that at some point these adc modules are put in the event stack in some order.

2. Defines the parameter names that will be assigned to each of the 16 channels of the module `adc1`. If you don’t want a channel to be assigned a parameter specify it’s name as "", and empty string. If the last several channels of a module will not be used, simply specify a shorter list.

   In this example, channel 0 of `adc1` will be called `ph1.00` and so on.

3. Similarly, the parameter names for the channels in the `adc2` module are defined.

I want to reiterate at this point that when `~/config/spectclconfig.tcl` runs it will actually create the parameters you specify as well as a 1-d spectrum for each raw parameter created.
set parameters(adc2) [list ph2.00 ph2.01 ph2.02 ph2.03 \
    ph2.04 ph2.05 ph2.06 ph2.07 \
    ph2.08 ph2.09 ph2.10 ph2.11 \
    ph2.12 ph2.13 ph2.14 ph2.15]

1. This section of the configuration file defines the modules
2. This section defines the events stack to read the two modules defined previously.
3. This section tells SpecTcl’s configuration scripts the parameter names to assign to each channel of the ADC modules. Note that spectra named ph1.00 ... ph1.15 and ph2.00 ... ph2.15 will be created that are 1-d spectra on the raw parameters.

**Notes**

1. Unlike C, C++ or Fortran, Tcl arrays are indexed by strings. Thus you can say things like:

   ```tcl
   set parameters(adcl) "a b c d e f g h i j k l m n o p"
   ``

   which creates an array element indexed by the string "adcl".
Chapter 4. SpecTcl

SpecTcl is the histogramming component of the NSCL data acquisition system. It uses an application called Xamine to display spectra that have been created.

SpecTcl is actually a library not a program. To make it work with a specific data set, the experimenter, in general must supply software that takes as input raw events, and produces as output parameters from which SpecTcl can increment the appropriate histograms.

For general applications, this user supplied software is organized as a logical pipeline of Event Processors. Event processors are run sequentially, each event processor has access to the raw event and to the parameters that have been computed by event processors that executed prior to it. Normally the first set of event processors operate on the raw event to produce raw parameters. Once this has been done, additional event processors can operate on the raw parameters to produce additional parameters without needing to know the form of the raw event.

An example of an event processor that might operate on decoded parameters would be a calibrator. A calibrator would take a set of raw parameters, and apply a calibration function to each of them to produce calibrated parameters (e.g. taking raw ADC values and producing energies).

The SpecTcl provided with this software includes a first stage event processor that knows how to unpack the raw events from the CC-USB for any set of stacks that can be defined by the daqconfig.tcl file we have described into a set of SpecTcl parameters.

This chapter describes how SpecTcl’s initialization scripts use the daqconfig.tcl definitions and variables to create parameter definitions and an initial set of spectra.

4.1. Using daqconfig.tcl to drive event decoding

SpecTcl needs to know two thing for each module:

- The order of appearance of each module in the event stack.
- The names to be given to parameters from each channel of a module.

The reference section of the Configuration Files chapter describes the parameters array. Recall that Tcl arrays are indexed by strings not numbers. Each element the parameters array should be a Tcl list that describes the names of the channels of a module. The index of each element is the name of a module created with the ph7xxx create command in the daqconfig.tcl file.
Chapter 4. SpecTcl

The **list** command is a Tcl command that will build lists with proper quoting and bracketing. Surround each list element that has spaces or {} brackets with quotes. Surround each list element that has [] brackets or $'s with {}'s. For example:

**Example 4-1. Using the Tcl list command**

```tcl
set parameters(someAdc) [list simple "has spaces or {}" {has [tcl special] $characters} ...
```

I strongly encourage you not to use Tcl special characters in parameter names, and to avoid spaces as well.

The config directory that is installed when you create a new account for data taking includes two scripts `daqconfig.tcl` and `spectclconfig.tcl`. `daqconfig.tcl` is a sample configuration script. `spectclconfig.tcl` is an initialization script for SpecTcl that knows how to process `daqconfig.tcl`.

SpecTcl can be told to use `spectclconfig.tcl` to process `daqconfig.tcl` when it starts.

Let’s conclude this section with an example of some code in the `SpecTclRC.tcl` initialization file that makes use of all this:

**Example 4-2. Incorporating automatic parameter/spectrum generation into SpecTcl**

```tcl
set setupFilename [file join ~ config spectclconfig.tcl]
source $setupFilename
```

This setup code is included in the `SpecTclRC.tcl` file that is installed in the data taking accounts you create.