A Domain Specific Language for defining Nuclear Physics Experiments.

R. Fox
National Superconducting Cyclotron Laboratory
Michigan State University
East Lansing, MI 48824-1321
K. Paräjärvi, J. Turunen, H. Toivonen
Säteilyturvakeskus
Helsinki, Finland

***Abstract*—Tcl's support for the implementation of domain specific languages (DSLs) has been used to define a language for describing the readout and initial unpacking of data from nuclear physics experiments. The DSL described has been tailored to the Weiner VM-USB USB/VME interface module, and the NSCLSpecTcl data analysis software. I will describe the extensions to TCL and a case study where this system was deployed at Säteilyturvakeskus (STUK) [1], the Radiation and Nuclear Safety Authority of Finland.**

I. Introduction

In this paper we will describe the implementation and deployment of a domain specific language for describing nuclear physics experiments. Interpretation of this language by extended Tcl interpreters provides for an end-to-end configuration of an experiment.

The presentation is divided into the following sections:

- Background - describes nuclear physics experimental techniques, how these techniques are crossing over into the field of environmental nuclear physics.
- Hardware and software structure
- Domain specific language structure and usage
- Results and Conclusions – describes the experienced of the Säteilyturvakeskus (STUK) Particles And Non Destructive Analysis (PANDA) group with the deployed system.

II. Background

To set the stage for the development of the DSL, we need two pieces of background. The first are the techniques of modern accelerator based nuclear physics experiments, the second, problems making measurements in the field of environmental nuclear physics. The discussion of these disparate fields will turn up a surprising opportunity for synergy that the PANDA group at STUK is in the process of exploiting.

A. Nuclear Physics Experimental Techniques

In accelerator-based experiments in nuclear physics, an accelerator produces a beam of ions moving with some energy. These ions bombard an experimental target where the some incident nuclei will interact with a nucleus in the target.

The result of each collision is a projectile fragment, and zero or more target fragments. Detectors arranged around the target provide measures, of the energy, mass and atomic number of these fragments. Computerized data acquisition systems acquire data from each event, and present them to applications that can record or analyze each collision online.

This simplified description glosses over the uncontrolled nature of these experiments. For each collision, several variables cannot be controlled:

- The incoming beam of accelerated ions is not mono-energetic.
- The incoming beam may not be pure.
- The distance between the centers of mass of the projectile and target nucleus (impact parameter) is random.
- The quantum mechanical nature of the nucleus implies that even given identical pre-conditions, the result is some probability distribution over the kinematically allowed results.

Physicists have become quite adept at using coincidence experiments, and event selection to analyze only the results of interesting interactions.

B. *Environmental Nuclear Physics*

Environmental Nuclear Physics seeks to understand, and quantify the hazards to people, and the ecology from the radionuclides in the environment. The Säteilyturvakeskus (STUK) is a Finnish governmental agency that, among other things, studies environmental nuclear physics in Finland and the surrounding regions.

STUK's work includes:

- Locating and quantifying presence of Radon gas due to the uranium rich geology of Finland.
- Monitoring the remaining traces of the Chernobyl plume of 1986.
- Checking for leaks and released waste from nuclear power plants.
- Contract work with former Soviet Union states to determine if dredging harbors is safe given that nobody knows where and how the Soviet Union disposed of spent naval nuclear reactor cores.
- Verification that nuclear power plants meet required construction standards.

The tool of choice for environmental nuclear physics is gamma ray spectroscopy. Fissionable nuclei decay by alpha emission to a daughter nucleus. After alpha decay, the daughter nucleus is in an excited state that rapidly decays to the ground state emitting a gamma ray with energy that is characteristic of the energy of the excited state. The energy of the gamma ray can uniquely identify daughter nucleus and therefore the parent.

Just as the description of accelerator-based experiments is simplified, the previous paragraph oversimplifies the process of analyzing a sample of air, earth or water for the presence of specific isotopes. Figure I. below shows the rate at which counts in a high purity Ge detector increment as a function of energy [2].

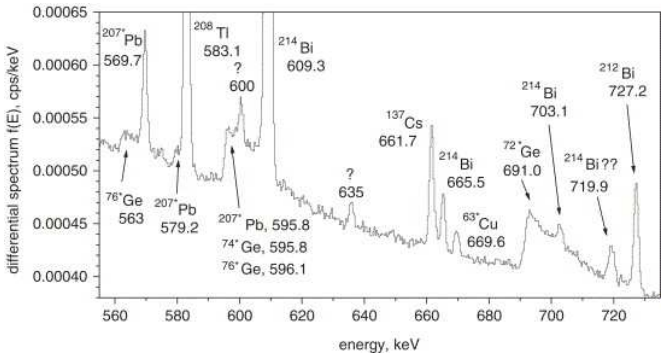

Figure 1 Natural gamma background spectrum

Due to the presence of naturally occurring radioactive elements isotopes, over 50 distinct peaks are observed in this spectrum, which sits on top of a large continuum of background. Since samples tend to be small, this natural background often obscures the peaks one is looking for. It is here that we can find a synergy between experimental nuclear physics and environmental nuclear physics.

In most cases we are interested in heavy elements. These decay by emitting an alpha particles as well as a characteristic gamma ray. The normal experimental nuclear physics technique would be to set up a pair of detectors: One sensitive to alpha particles and the other sensitive to gammas. In analyzing the data, or at the electronics trigger level, we would require a gamma to be in time coincidence (in the same decay event), as an alpha.

Figure 2 below shows the results of tests STUK did at the nuclear research lab in Jyväskylä [3]. Alpha-Gamma coincidence events were acquired from the "Thule" particle, a 25μm diameter particle that is thought to be a piece of weapons grade material. Note how the background is completely removed by requiring the coincidence.

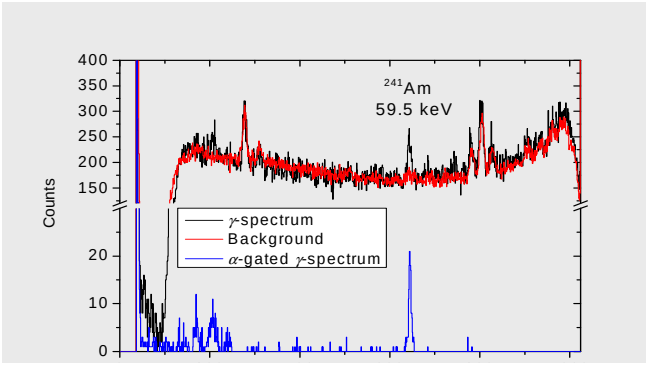

Figure 2 Thule particle spectrum

The two (possibly three) peaks that remain come from (right to left) the decays of 241Am and 235U and 239Pu. The Uranium and Plutonium peaks are present because 239Pu is the weapons grade material of choice and 235U is a daughter product of the decay of 239Pu. 241Am is produced as an unintended by-product of the process that produces 239Pu.

III. HARDWARE AND SOFTWARE

On the basis of its tests at Jyväskylä, STUK contracted to have a dedicated data taking system constructed. They chose to base this system on the open sourced NSCL data acquisition

system[4] and the NSCLSpecTcl [5] online analysis and display system. While the initial contract specified data taking from a 32x32 double sided silicon strip detector (DSSSD) and a High Purity Germanium (HPGe) detector, the counting chamber they constructed was considerably more flexible. The contract therefore specified a system that was expandable and upgradeable by STUK personnel.

Figure 3 shows the STUK PANDA (Particles And Non Destructive Analysis) counting chamber. An airlock allows the sample holder to be removed from the system without losing vacuum in the counting area. The chamber contains two counting areas. Counting area 1 is initially instrumented with a DSSSD detector and an HPGe detector. The HPGe detector is the detector of choice for high precision gamma ray spectroscopy. The DSSSD detector is charged particle sensitive, and consists of 32 vertical and 32 horizontal segments. The subdivision of the detector allows for a reconstruction of the position of the sample in a sample holder. The bag of Panda brand licorice on the counting chamber gives a sense for the size of the chamber.

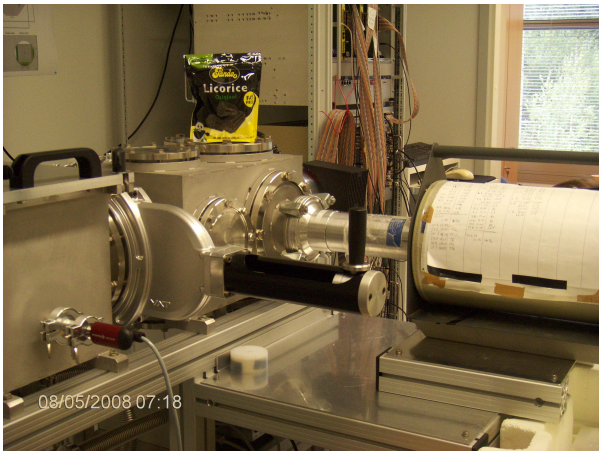

Figure 3 The STUK PANDA setup

Counting area 2 is not yet populated. However one idea for future expansion would be to do a coarse determination of the position of the active particle in the sample holder in position 1 and then moving the sample to counting area 2 for a more precise position identification or more precise alpha energy determination. Future developments could include the deployment of other charged particle detectors in counting area 1, such as micro channel plate detectors or pixilated silicon detectors.

The electronics used to take data from this system is shown in block diagram form in Figure 4:

- The Mesytec, MADC32 is a new peak sensing ADC capable of digitizing peak heights from signals that have rise times and durations typical of those seen in nuclear particle detectors.
- The VM-USB is an FPGA based intelligent readout system It is capable of sub-microsecond trigger latencies and can initiate a VME-bus transfer every 230ns. It transfers data to a host computer at full USB-2 data rates.

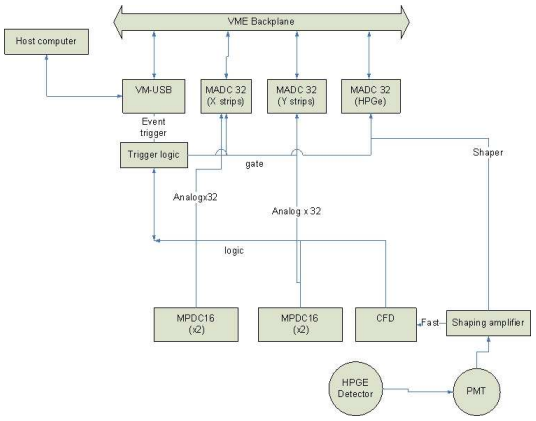

Figure 4 STUK electronics block diagram

Software is the standard NSCL data acquisition system with the readout component tailored for the VM-USB. See the block diagram in Figure 5 below for the structure of this software.

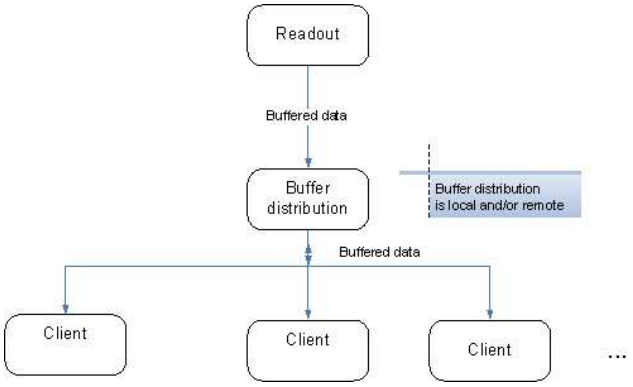

Figure 5 Software block diagram

IV. The Domain Specific Language

We chose to use a domain specific language to configure the readout program and NSCLSpecTcl's initial unpack of the raw event data. A domain specific language allows users who are not expert hardware programmers to easily select and configure the set of digitizers they use and which ones to read in response

to a trigger. Using the DSL description of the experiment to drive NSCLSpecTcl's event unpacking ensures consistency between the readout software and the analysis software.

In this section several aspects of the DSL will be described:

- The general form of the language
- The Readout program's configuration parameter subsystem.
- How the Readout program uses the language
- How NSCLSpecTcl interprets the DSL configuration.

A. Form of the DSL

The DSL we have created and used describes how to configure VME based electronics, and how the VM-USB should respond to triggers. A command ensemble represents each supported electronics module type. While the STUK system is now entirely composed of Mesytec MADC-32 ADCs, command ensembles that support many other digitizers in use in nuclear experimental physics have also been implemented. Each command ensemble implements a minimum of three standard subcommands:

- **create** - creates a new instance of this module type and supplies a name and an optional configuration string
- **config** - further configures an existing instance of a module type.
- **cget** - queries the configuration of a module type as a set of configuration option/value pairs.

The configuration of a module instance is a set of option/value pairs. Each module type declares the set of options it supports. The resemblance to Tk is intentional. Where Tk creates and configures 'virtual objects' that have screen presence, our DSL creates and configures software proxies that control actual hardware physically present in the system.

The VM-USB is capable of responding to up to 8 trigger conditions. These include front panel signals, a repeating timer, and VME backplane interrupts. In response to a trigger, the VM-USB executes a set of pre-programmed VME operations called a *stack*. Stacks can contain arbitrary VME reads and writes as well as block transfers. Write data is inline while data read is buffered into the event that corresponds to the trigger. Blocks of events are delivered to the host computer over the USB-2 interface.

The VM-USB is modeled as an electronics module class that can provide stack instances. Stack instances can be configured just like any other electronics module. They have a -trigger option that specifies the trigger source, and -modules option that is a list of other non-stack modules that will be read in response to the stack instance's trigger.

Here is a segment of the initialization file used for the STUK alpha-gamma coincidence measurements:

```
madc create dsssd1.y  -base 0x50000000 -id 5 -ipl 0
madc config dsssd1.y  -gatemode common \
                      -gategenerator disabled
madc config dsssd1.y  -inputrange 8v
madc config dsssd1.y  -timestamp on -timingsource vme \
                      -timingdivisor $madcTimeDivisor
madc config dsssd1.y  -thresholds $thresholds
...
stack create events
stack config events -trigger nim1 \
            -modules [list dsssd1.x dsssd1.y hpge ]  \
            -delay 15
```

Example 0 Configuration file segment

The first set of **madc** commands create and configure the MADC32 instance named dsssd1.y. The vertical strips (y orientation), will be digitized by this ADC. The -thresholds option takes a Tcl list of 32 channel thresholds. If any channel's signal height would convert below its threshold value, the data from that channel is suppressed from the event's output data.

The stack command creates and configures the events stack. This stack is used to respond to the primary event trigger. The primary event trigger is configured to be the VM-USB NIM1 input. The list of modules to read is the value of the -modules switch. A 15-microsecond delay between trigger and stack execution is configured to allow the modules to convert their input signals before their readout begins.

B. Configuration Parameter Subsystem

The create subcommand of a device command in the DSL will create an instance of a device driver class. The management of the configuration database is a significant part of what a physical device driver module must do. Configuration management has been factored into a subsystem dedicated to processing, validating, and storing and fetching configuration parameters.

When a device driver instance is created, it is responsible for registering its configuration options. Configuration option values are strongly typed and in many cases additional constraints may apply. For example, the madc32 –threshold value must be a list of exactly 32 integers and each integer must lie in the range 0 through 4095.

When the device driver registers a parameter option with its configuration database, it can specify an optional validation function. The validation function, or validator, verifies that a proposed configuration option value is legal. An optional

validator parameter allows value constraints to be passed to the validator.

The configuration subsystem supplies validators for the most commonly used data types. These stock validators include a flexible set of constraint arguments that cover most needs. For example, the list validator constraint parameter is a structure that allows lower and upper limits to be set on the list size as well as an additional validator function and parameter to be applied to each element of the list. Configuration parameter values are always stored as strings.

Device driver instances must query the device database when programming the hardware. The configuration subsystem provides functions that allow drivers to fetch configuration parameters from their configuration databases. While configuration parameter values are stored as strings, the configuration parameter database subsystem provides member functions that allow a parameter to be fetched and converted into the most common data types as well.

C. Readout's Use of the DSL

Each supported electronics module consists of two C++ classes. The first class is the module device driver. Instances of the module device driver are responsible for:

- Describing a set of configuration parameters, their validators and constraints.
- Preparing a hardware device for data taking in accordance with its configuration database settings.
- Contributing elements to the stacks to which this instance was added.

The second class is the module's command ensemble. The command ensemble is implemented using the template pattern. Specific command ensemble classes need only provide a function that knows how to create a driver instance for their module type.

At the beginning of each run Readout;

- Forgets all existing module device driver instances.
- Constructs a new Tcl interpreter to process the configuration DSL script.
- Gets each module device driver to register its command ensemble with the interpreter
- Processes the script.
- Initializes appropriate modules, and stacks.

When the configuration script is processed, it will, in invoke commands in the DSL. The **create** subcommand of each DSL command ensemble creates a new device driver instance, attaches a configuration database to that instance. Readout then asks the driver instance to declare its configuration parameters to its configuration database instance.

The **config** command locates the appropriate device driver instance, obtains its configuration database and passes configuration value/option pairs to the driver instance reporting any errors. This command can operate without involving the device driver instance in any way, unless the driver has provided a custom validator.

The **cget** subcommand locates the driver instance, and its configuration database, and produces a dump of that database as a Tcl list of name/value pairs as the command result. **cget** provides the configuration script with the ability to introspect the properties of known modules.

Any errors detected when processing the configuration script result in an informative error message. Errors cancel data taking.

When the configuration script has been processed without errors, the hardware is initialized, and stacks loaded into the VM-USB. The VM-USB must be programmed respond to triggers. The stacks defined in the DSL description drive the hardware initialization and stack generation. The assumption is that there's no point in initializing hardware that does not participate in data taking.

When a driver instance is created it is stored with two lookup keys. The first key is the name of the instance (the name on the create subcommand). The second key is the class of device (provided by the creator). The Readout software gets a list of all of the driver instances of class *stack*. These are instances of VM-USB stack drivers. The stacks are asked to initialize themselves. They do this by locating the driver instance for each module in their -module configuration option list and asking each one to initialize itself in turn. The driver instance initialization member functions interrogate their configuration database and interact with the VM-USB to setup the hardware as desired. At this time the stack also does whatever stack specific initialization it must do such as setting the trigger condition for the stack.

Once the hardware has been initialized, the stack module instances take another pass through their module lists. Each module instance is asked to contribute instructions to a data structure that is a direct representation of a VM-USB stack. These stacks are then loaded into the VM-USB, and VM-USB trigger processing is enabled.

D. NSCLSpecTcl's use of the DSL

NSCLSpecTcl is a general-purpose histogramming package for nuclear physics event data. NSCLSpecTcl has been described in detail in a previous Tcl conference [5]. Figure 6 below

shows the flow of data through NSCLSpecTcl and highlights the subset of SpecTcl that must manage the domain specific experiment description file.

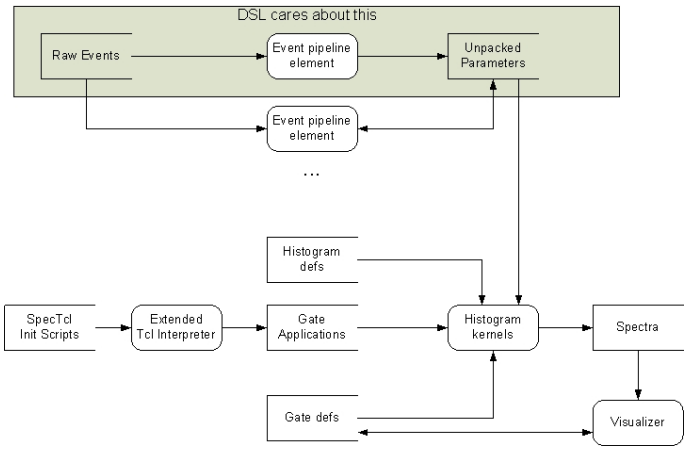

Figure 6 SpecTcl Dataflow

NSCLSpecTcl's first stage of analysis decodes raw events to a set of parameters. Normally this decode is done by hand written experiment specific code. Since the format of an event is fully determined by the order and types of the modules read out, using a DSL to specify the experiment enables the use of a general-purpose event decoder that is driven by that description.

NSCLSpecTcl needs only a subset of the information supplied by the module creation and configuration commands. Specifically, for each module, it must know the type of that module. For each stack, it must know the stack number (determined by the trigger configuration) and the ordered list of modules in that stack.

Some further metadata must be supplied, however. NSCLSpecTcl needs to know the names of the parameters into which the channels of each electronics module must be unpacked. The user supplies this data via a Tcl array. The Tcl array is indexed by module name. Its elements are the list of the names of parameters the named module has. In early versions of the DSL, each module device driver provided a -parameters option, which was ignored by the device driver and processed by NSCLSpecTcl. Users, however think of the electronics configuration and parameter naming as separate processes and therefore reacted better to this scheme.

Two commands **stackmap** and **parammap** have been added to NSCLSpecTcl. **stackmap** defines, for each stack the order and type of the modules read by that stack. **parammap** defines the set of parameters each module supplied SpecTcl. A pure Tcl script interprets the experiment description file for NSCLSpecTcl , issues the appropriate **stackmap** and **parammap c**ommands, and even creates an initial set of spectra for all of the raw parameters.

NSCLSpecTcl DSL script interpretation is broken into two components. *configFile.tcl* interprets the DSL creating global arrays that are an equivalent description of the experiment. A second script, *spectclSetup.tcl* operates on this global data to issue the correct set of **parammap** and **stackmap** commands along with the appropriate NSCLSpecTcl commands.

Separating the parsing of the configuration file from issuing the resulting NSCLSpecTcl commands allows the *configFile.tcl* package to be re-used. The STUK contract specified the delivery of a conversion program from NSCL Event file format to an equivalent XML representation. In writing this program we were able to re-use *configFile.tcl* as well as the NSCLSpecTcl module unpackers.

V. Results and Conclusions

Development of the software for STUK contract was was completed in late May 2008. Acceptance testing with pulsers was done at the NSCL in June 2008 and required very little tweaking. An installation/training visit to STUK was arranged for late July. R. Fox spent an initial 5 days in Helsinki doing installation and training. This was followed by nine days for the STUK staff to use the system by themselves (while R. Fox junketed about Helsinki, Estonia and Germany) Finally, R. Fox returned to STUK for two days of follow up.

The system was assembled, connected to the detectors and taking data from a triple alpha source within the first 1/2-day. As STUK had not yet received their HPGe detector we were unable to look at gamma ray spectra. The next few days demonstrated several issues with the Mesytec MADC32, which fortunately could be resolved via e-mailed firmware updates, applied while at STUK. This trouble shooting often involved re-cabling the DSSSD to different ADC channels, which, in turn changed the position mapping between ADC, channel and detector position. The DSL and metadata approach allowed us to change the configuration file almost as quickly as we changed cabling.

With the ADC issues resolved, R. Fox provided about 2 or three hours of training on configuration script writing with the DSL. One of the STUK staff members involved with the project, K. Peräjärvi, had some prior experience with Tcl. Following the training, the STUK staff was presented with the following set of real-world exercises:

1. Re-cable the DSSSD back to its original configuration and adjust the channel/parameter map appropriately.
2. Add a pulser channel that would simulate a high-resolution silicon detector in station 2.

3. Determine and set the MADC32 thresholds to suppress channels without valid conversions.

The STUK users easily handled these problems.

At the end of the first week of onsite support, the STUK staff was able to borrow an HPGe detector from Jyväskylä that could be used with their chamber. In their week of unsupported use they took data from the Thule particle described in section II and were easily able to reproduce the background subtracted spectrum from their Jyväskylä test run. Additional run time allowed them to clearly see the 239Pu peak that was only tentatively identified during the Jyväskylä tests.

Throughout the installation and training, members of other groups at STUK often stopped by the lab to see the system. All were excited about the capabilities of the system, and at how easy it was to reconfigure the software via the DSL. Many speculated on other applications for the system.

The European Atomic Energy Commission EURATOM is closely following the work at STUK with the idea that the system could be distributed to safety groups in other member countries.

For the most part we are satisfied with the DSL. The main weakness is the need to modify the source code of the compiled readout application to add support for a new device type to the system. If I we were starting again, it might be a better choice to expose lower level VM-USB primitives to the Readout interpreter and then build the device driver software on top of that in pure Tcl (e.g. as snit::type's).

This approach is possible for the Readout software, because device driver software is only involved at the start of data taking, and therefore not speed critical. NSCLSpecTcl is a different matter, however. The device unpacking software lives in the innermost innermost loop of software that must process literally millions of events. NSCLSpecTcl does not have to interact with the hardware and is therefore much simpler software for members of the physics community to extend.

VI. References

[1] http://www.stuk.fi

[2] A very long-term HPGe-background gamma spectrum P Bossew Applied Radiation and Isotopes 62 (2005) 635-644

[3] K. Peräjärvi et al. in publication process for IEEE TNS and private communication with J. Turunen (STUK).

[4] Real-Time Results Without Real-Time Systems R. Fox et al. IEEE TNS Vol 51. No. 3 June 2004 571-575

[5] SpecTcl: A Nuclear Physics Data Analysis package Ron Fox Tcl 11'th Annual Tcl Conference New Orleans, LA http://www.tcl.tk/community/tcl2004/Papers/RonFox/fox.pdf