The MapValues plugin

Ron Fox

The MapValues plugin

by Ron Fox

Revision History

Revision 1.0 October 3, 2008 Revised by: RF
Original Release

Table of Contents

List of Examples

Chapter 1. Introduction

This document describes the MapValues SpecTcl plugin. The plugin allows you to create mappings from one set of parameter values to another that are executed at compiled speed.

One sample use case might be a strip detector. Suppose that you want to create a position parameter. This could be generated from the strip number of the strip with the highest conversion value. On looking at the manual for the cable, you discover, however that the cable pins don't provide the strips in order, and that you'll have to unscramble them to get a strip number, which can then be converted into a position.

You could code this as a table lookup from strip position to final geometric position, or you could just generate the position parameter fromthe cable pin number using the MapValues plugin.

The remainder of this document is organized as follows:

- The next chapter describes how to get and install the plugin.
- Chapter 3 describes how to load and use the plugin.
- Chapter 4 provides a command reference for the plugin.

In the sample dialogs with the computer, computer output is shown in `this font` while user input **`looks like this`**.

Chapter 2. Obtaining and installing the plugin

The plugin is available from http://www.sourceforge.net/projects/nsclspectcl To download click "Download" at the top of the page. Selecdt the mapvalues tarball and download it.

Once downloaded, the tarball must be unpacked and the software built. To unpack the software for version 1.0 e.g.:

```
$ tar xzf mapvalues-1.0.tar.gz
```

The tarball will create a new subdirectory that is the name of the tarball, without the .tar.gz.

To build the software, cd to that directory, configure and make the software. The sample below assumes that you are installing the software to `/usr/opt/spectcl/3.3` to supply plugins for SpecTcl 3.3. The actual software will be installed inthe `TclLibs/mapvalues` subdirectory of the specified installation direcotory. This is done to allow more than one plugin to be sintalled in the same configured destination.

```
$ cd mapvalues-1.0
$ ./configure --pefix=/usr/opt/spectcl/3.3
 ...
$ make install
$  make mapvalues.pdf  #docs
```

If you are not installing the plugin in the same directory tree SpecTcl is installed in, you must specify where the SpecTcl location is by using the `--with-spectcl-home` option on the **configure** command line.

The `--help` switch makes **configure** print a list of options and parameters it supports.

Chapter 3. Loading and using the plugin

The plugin is a compiled loadable Tcl package. If it is installed in the SpecTcl package search path, it can be loaded via the **package require** command directly. If not, you should add the `TclLibs` subdirectory of the installation directory either to the environment variable `TCLLIBPATH` or to the Tcl list `auto_path`.

Once the installation is in the path, you can load the plugin via the command:

```
%  package require MapValues
1.0
%
```

Once the plugin has been installed, it can be used to create new parameters from old parameters. See Value map plugin command reference for a complete description of the command added by the plugin.

In this chapter we'll content ourselves with presenting some examples.

The first example shows how to reverse the strips of a 32 strip silicon strip detector. and create a spectrum from that new parameter. For the sake of this example, the raw strip number will be the parameter named `dsssd1.stripnumber`. The resulting parameter will be named `dsssd1.position`.

Example 3-1. Reversing strip numbers for 32 strips.

```
package require MapValues
# Produce the mapping list

for {set i 0} {$i < 32} {incr i} {
   set mappedValue [expr 31-$i]
   lappend mapping [list $i $mappedValue]
}
# Create the new parameter and spectrum from it:

mapvalue dsssd1.stripnumber dsssd1.position $mapping
spectrum dsssd1.position 1 dsssd1.position {{0 31 32}}
```

The second (and last) example is an extension of the first. It assumes the detector strips are 2mm wide and produces an actual position parameter. Note that in this case the parameter was defined in advance so that units could be associated with it.

Example 3-2. Mapping strip numbers to positions.

```
package require MapValues

set stripSpacing 2.0;      # mm between strips.
```

```
for {set i 0} {$i < 32} {incr i} {
   set mappedValue [expr (31-$i)*$stripSpacing]
   lappend mapping [list $i $mappedValue]
}
# Create the new parameter and spectrum from it:

set low 0.0
set high [expr 31.0*$stripSpacing]
treeparameter create dsssd1.position $low $high mm
mapvalue dsssd1.stripnumber dsssd1.position $mapping
spectrum dsssd1.position 1 dsssd1.position "{[list $low $high 32]}"
```

Chapter 4. Value map plugin command reference

mapvalue

Name

`mapvalue` — Create new parameters via simple maps of existing parameters

Synopsis

package require MapValues

mapvalue inParam outParam mappingList

DESCRIPTION

This command creates and installs an event processor that computes *`outParam`* from *`inParam`* on an event by event basis by applying a simple table driven transformation. *`mappingList`* specifies this transformation. Once this new parameter is defined, it can be treated just like any other SpecTcl parameter (histogrammed, have gates set on it etc.).

The new parameter *`inParam`* is created if not yet defined to SpecTcl, by assigning an unused parameter slot. If the parameter already exists, it will be used as defined. Later plugins can locate this parameter via the SpecTcl API. (This implies that a mapping of a parameter defined by a **mapvalue** command can in turn also be the input parameter of a new **mapvalue** command.

The mapping is a discrete, lookup-based mapping defined by the *`mappingList`* parameter. As the name implies, this is a Tcl list. Each list element is a two element sublist. The first sublist element is a value of the input parameter. The second sublist element, is the resulting value of the output parameter. The input value must be an integer. The output value is interpreted as a floating point value.

EXAMPLES

Examples are given in the chapter Loading and Using the Plugin.