const parameter plugin

Ron Fox

const parameter plugin

by Ron Fox

Revision History

Revision 1.0 August 27, 2009 Revised by: RF
Original Release

Table of Contents

List of Examples

Chapter 1. Introduction and installation.

The const plugin allows you to create a new SpecTcl parameter that has a constant value if either one of a set of parameters is defined or all of a set of parameters is defined in an event. The plugin adds the **const** command to SpecTcl, as well as an event processor that computes the parameters created by the **const** command.

The const plugin facilitates the production of rate/stripchart spectra. Suppose you already have a parameter that represents time. You want to know the rate at which you are getting at least one hit in a set of detectors. You can create a const parameter based on some parameter that indicates one of those detectors is hit (such as a timing parameter), and plot that const parameter against your time in a stripchart spectrum.

The remainder of this chapter describges:

1. How to obtain the const parameter plugin.
2. How to install the plugin

The second chapter Using the const plugin describes how to use the plugin and provides a reference page for the **const** command.

1.1. Obtaining the plugin.

The plugin can be obtained from the SourceForge NSCL SpecTcl project site (http://www.sourceforge.net/projects/nsclspectcl). As Sourceforge has been changing the appearance of the site precise click by click instructions are not possible. What can be said is that the downloads are organized by package. Each package has a set of releases, and each release has a set of files. All plugins are in the plugins package. Go to the latest release and download the file with a name that begins `constparam`. The filename will be of the form `constparam-`*`a.b-nnn`*`.tar.gz`. Where *`a.b-nnn`* will be the plug in version number.

1.2. Installing the plugin

We will assume that you have downloaded a tarball named `constparam-1.0-001.tar.gz` in the sample commands below. Substitute the actual name of the tarball where appropriate in the example commands.

Unwrap the tarball into a new directory:

```
tar xzf constparam-1.0-001.tar.gz
```

This should create a new directory tree under your current working director named `constparam-1.0-001`. enter that directory

```
cd constparam-1.0-001
```

Plugin installation follows the 'usual' GNU software approach of running a configuration script and then using the Unix **make** command to build and install the software.

The plugin configuration will need to know several things. These can be provided via a set of command line switches to the **configure** script, defaulted or, in some cases searched for by the script:

`--prefix=`*`path`*

Describes the top level directory in which the plugin should be installed. The actual plugin files will be installed in the `TclLibs` directory within that *`path`*.

Normally plugins are installed in the installation directory tree of a specific SpecTcl installation. For example at the NSCL, for SpecTcl 3.3, *`path`* is `/usr/opt/spectcl/3.3`. see the `--with-spectcl-home` switch below, however.

`--with-spectcl-home=`*`path`*

Provides the top leve of the SpecTcl installation directory. This must be where a specific version of SpecTcl is installed. If not provided, this defaults to the value of the `--prefix` option.

`--with-tcl-header-dir=`*`path`*

In most cases, the configuration script is able to locate the Tcl C API headers. If, however this search fails, you can explicitly point the configuration at the directory that contains `tcl.h` by using the `--with-tcl-header-dir` switch.

The **configure** script provides quite a few additional. switches and listens to several environment variables. To get a listing of these invoke the script with the `--help` switch.

The sample line below configures the plugin to be installed in the `TclLibs` directory tree of SpecTcl-3.2 at the NSCL:

```
./configure --prefix=/usr/opt/spectcl/3.2
```

Once configured, you use **make** to build and install the software:

```
make install
```

The example below pulls all of this together to show a typical installation command set.

Example 1-1. Installation example

```
tar xzf constparam-1.0-001.tar.gz
cd constparam
./configure --prefix=/usr/opt/spectcl/3.2
make install
```

Chapter 2. Using the const plugin

This chapter describes how to use the const plugin within your tailored SpecTcl. We will assume that you've installed the plugin in the SpecTcl installation directory tree, however it should be clear how to modify these instructions if you've installed the plugin elsewhere.

This chapter specifically contains:

1. Instructions for loading the plugin into a running SpecTcl
2. Instructions for createing and deleting const parameters.
3. Reference information for the **const** command created by the plugin.

2.1. Loading the plugin

SpecTcl plugins look like Tcl loadable packages to SpecTcl. To load a plugin you therefore must ensure that the directory tree containing the plugin is part of the Tcl `auto_path` search list global variable.

Again, assuming you are using SpecTcl 3.2, as installed at the NSCL, you can do this by adding the following line to your `SpecTclRC.tcl` intialization file:

```
lappend auto_path /usr/opt/spectcl/3.2/TclLibs
```

The Tcl **package** command can then be used to load the plugin by name. Once more the command below can be added to your `SpecTclRC.tcl` initialization script:

```
package require constparam
```

You can verify interactively that the package loaded successfully by asking Tcl if it knows about the const command.

```
info commands const
```

If the package loaded successfully, you Tcl will echo back `const`. If not, Tcl will echo back an empty result. If the load has failed, try loading the package interactively and see if the error messages issued help you figure out what went wrong.

2.2. Using the plugin

The plugin add the **const** command to SpecTcl. You can use this command to create and delete parameters managed by the plugin. Parameters created by the **const** command are just like any other SpecTcl parameter, gates can depend on them, Spectra can be made from them, psuedo parameters can bge defined that depend on them and so on.

However, in order to ensure that all parameters required by a const parameter have been defined or not when the const parameters are computed, each time you define a const parameter, the event processor responsible for computing all const parameters is shifted to the end of the event processor pipeline.

Const parameters depend on a list of existing parameters. There are two types of const parameters. The default type is an `or` parameter. It is defined if any of the parameters it depends on has been given a value in that event. You can also define an `and` parameter which is only defined if *all* of the parameters it depends on have been given a value.

The commands below define first an or and then an and parameter:

```
const anorparameter 1 [list p1 p2 p3 p4]
const -and anandparameter 1 [list p5 p6 p7]
```

As you can see, the only difference between the form of the two commands is the presence of the `-and` option on the second, and parameter definition.

The command line parameters are:

1. The name of the parameter to be created
2. The value to be assigned the new parameter if it should be defined.
3. The list of parameters the new parameter depends on.

While the SpecTcl **parameter -delete** command can be used to delete the definition of a const parameter, the event processor that computes these parameters will not be aware of that fact. You should therefore always delete (if necessary) const parameters via the **const** command.

The sample below deletes the and parameter we created:

```
const -delete anandparameter
```

2.3. The const command reference

const

Name

`const` — Create/Delete const parameters

Synopsis

const ?-and? outparam outvalue list-of-input-parameters

const -delete outparam

DESCRIPTION

The **const** command is added to the set of SpecTcl commands by loading the const plugin. It allows you to create or delete constant parameters. A constant parameter is one that is conditionally assigned a constant value depending on the presence or absence of previously define SpecTcl parameters.

Two types of constant paramters can be defined. `Or` parameters are defined if at least one of the parameters they depend on were given values. `And` parameters are only defined if all of the parameters they depend on have been assigned values.

The first form of the command in the SYNOPSIS section creates a new parameter. If the `-and` option is present, an `And` parameter is created as described in the previous paragraph. The new parameter will be named *`outparam`*. When defined it will be given the value *`outvalue`* which must evaluate to a floating point constant. *`list-of-input-parameters`* is a properly formatted Tcl list of existing SpecTcl parameter names that provide the parameters required to define the *`outparam`* for each event. You can use the Tcl **list** command to generate this list.

The second form of the parameter deletes the constant parameter named *`outparam`*. No action will be taken, and an error will be emitted if *`outparam`* is not a constant parameter (created by *`const`*).

OPTIONS

`-and`

Used when creating a new constant parameter to require that all the parameters in the *`list-of-input-parameters`* must be given a value in order to give a value to the output

parameter.

`-delete`

Used after the **const** command to indicate the command is deleting the *`outparam`*

EXAMPLES

Example 2-1. Creating or and and parameters

```
const anorparameter 1 [list p1 p2 p3 p4]
const -and anandparameter 1 [list p5 p6 p7]
```

Example 2-2. Deleting an existing const parameter

```
const -delete anorparameter
```