Snit's Not Incr Tcl README.txt

-----------------------------------------------------------------

Snit is pure-Tcl object and megawidget framework. See snit.html

for full details.

Snit is part of "tcllib", the standard Tcl Library.

Snit lives in "tcllib" now, but it is available separately at

http://www.wjduquette.com/snit. If you have any questions, bug

reports, suggestions, or comments, feel free to contact me, Will

Duquette, at will@wjduquette.com; or, join the Snit mailing list (see

http://www.wjduquette.com/snit for details).

Differences Between Snit 2.1 and Snit 1.x

--------------------------------------------------------------------

V2.0 and V1.x are being developed in parallel.

Version 2.1 takes advantage of some new Tcl/Tk 8.5 commands

([dict], [namespace ensemble], and [namespace upvar]) to improve

Snit's run-time efficiency. Otherwise, it's intended to be

feature-equivalent with V1.x. When running with Tcl/Tk 8.5, both

V2.0 and V1.x are available; when running with Tcl/Tk 8.3 or Tcl/Tk

8.4, only V1.x is available.

Snit 1.x is implemented in snit.tcl; Snit 2.1 in snit2.tcl.

V2.1 includes the following enhancements over V1.x:

\* A type's code (methods, type methods, etc.) can now call commands

from the type's parent namespace without qualifying or importing

them, i.e., type ::parentns::mytype's code can call

::parentns::someproc as just "someproc".

This is extremely useful when a type is defined as part of a larger

package, and shares a parent namespace with the rest of the package;

it means that the type can call other commands defined by the

package without any extra work.

This feature depends on the new Tcl 8.5 [namespace path] command,

which is why it hasn't been implemented for V1.x. V1.x code can

achieve something similar by placing

namespace import [namespace parent]::\*

in a type constructor. This is less useful, however, as it picks up

only those commands which have already been exported by the parent

namespace at the time the type is defined.

There are four incompatibilities between V2.1 and V1.x:

\* Implicit naming of objects now only works if you set

pragma -hastypemethods 0

in the type definition. Otherwise,

set obj [mytype]

will fail; you must use

set obj [mytype %AUTO%]

\* In Snit 1.x and earlier, hierarchical methods and type methods

could be called in two ways:

snit::type mytype {

method {foo bar} {} { puts "Foobar!"}

}

set obj [mytype %AUTO%]

$obj foo bar ;# This is the first way

$obj {foo bar} ;# This is the second way

In Snit 2.1, the second way no longer works.

\* In Snit 1.x and earlier, [$obj info methods] and

[$obj info typemethods] returned a complete list of all known

hierarchical methods. In the example just above, for example,

the list returned by [$obj info methods] would include

"foo bar". In Snit 2.1, only the first word of a hierarchical

method name is returned, [$obj info methods] would include

"foo" but not "foo bar".

\* Because a type's code (methods, type methods, etc.) can now

call commands from the type's parent namespace without qualifying

or importing them, this means that all commands defined in the

parent namespace are visible--and can shadow commands defined

in the global namespace, including the standard Tcl commands.

There was a case in Tcllib where the Snit type ::tie::std::file

contained a bug with Snit 2.1 because the type's own name

shadowed the standard [file] command in the type's own code.

Changes in V1.2

--------------------------------------------------------------------

\* Defined a family of validation types. Validation types are used

to validate data values; for example, snit::integer and its

subtypes can validate a variety of classes of integer value, e.g.,

integers between 3 and 9 or integers greater than 0.

Changes in V1.1

--------------------------------------------------------------------

\* It's now explicitly an error to call an object's "destroy" method

in the object's constructor. (If you need to do it, just throw

an error; construction will fail and the object will be cleaned

up.

\* The Tile "ttk::frame" widget is now a valid hulltype for

snit::widgets. Any widget with a -class option can be used

as a hulltype; lappend the widget name to

snit::hulltypes to enable its use as a hulltype.

\* The TK labelframe widget and the Tile ttk::labelframe widget are

now valid hulltypes for snit::widgets.

Changes in V1.0

--------------------------------------------------------------------

Functionally, V1.0 is identical to version V0.97.

\* Added a number of speed optimizations provided by Jeff Hobbs.

(Thanks, Jeff!)

\* Returned to the name "Snit's Not Incr Tcl".

\* Fixed SourceForge Tcllib Bug 1161779; it's no longer an error

if the destructor is defined before the constructor.

\* Fixed SourceForge Tcllib Bug 1106375; the hull widget is now

destroyed properly if there's an error in the constructor of

a widget or widgetadaptor.

Changes in V0.97

--------------------------------------------------------------------

The changes listed here were actually made over time in Snit V0.96;

now that they are complete, the result has been renumbered Snit V0.97.

\* Bug fix: methods called via [mymethod] can now return exotic

return codes (e.g., "return -code break").

\* Added the -hasinfo pragma, which controls whether there's an

"info" instance method or not. By default, there is.

\* POSSIBLE INCOMPATIBILITY: If no options are defined for a type, neither

locally nor delegated, then Snit will not define the "configure",

"configurelist", and "cget" instance methods or the "options"

instance variable.

\* If a snit::type's command is called without arguments, AND the type

can have instances, then an instance is created using %AUTO% to

create its name. E.g., the following commands are all equivalent:

snit::type dog { ... }

set mydog [dog create %AUTO%]

set mydog [dog %AUTO%]

set mydog [dog]

This doesn't work for widgets, for obvious reasons.

\* Added pragma -hastypemethods. If its value is "yes" (the

default), then the type has traditional Snit behavior with

respect to typemethods. If its value is "no", then the type

has no typemethods (even if typemethods were included

explicitly in the type definition). Instead, the first argument

of the type proc is the name of the object to create. As above,

the first argument defaults to "%AUTO%" for snit::types but not

for snit::widgets.

\* Added pragma -simpledispatch. This pragma is intended to make

simple, heavily used types (e.g. stacks or queues) more efficient.

If its value is "no" (the default), then the type has traditional

Snit behavior with respect to method dispatch. If its value is

"yes", then a simpler, faster scheme is used; however, there are

corresponding limitations. See the man page for details.

\* Bug fix: the "pragma" statement now throws an error if the specified

pragma isn't defined, e.g., "pragma -boguspragma yes" is now an

error.

\* Bug fix: -readonly options weren't. Now they are.

\* Added support for hierarchical methods, like the Tk text widget's

tag, mark, and image methods. You define the methods like so:

method {tag add} {args} {...}

method {tag configure} {args} {...}

method {tag cget} {args} {...}

and call them like so:

$widget tag add ....

The "delegate method" statement also supports hierarchical methods.

However, hierarchical methods cannot be used with -simpledispatch.

\* Similarly, added support for hierarchical typemethods.

Changes in V0.96

--------------------------------------------------------------------

V0.96 was the development version in which most of the V0.97 changes

were implemented. The name was changed to V0.97 when the changes

were complete, so that the contents of V0.97 will be stable.

Changes in V0.95

--------------------------------------------------------------------

The changes listed here were actually made over time in Snit V0.94;

now that they are complete, the result has been renumbered Snit V0.95.

\* Snit method invocation (both local and delegated) has been

optimized by the addition of a "method cache". The primary

remaining cost in method invocation is the cost of declaring

instance variables.

\* Snit typemethod invocation now also uses a cache.

\* Added the "myproc" command, which parallels "mymethod". "codename"

is now deprecated.

\* Added the "mytypemethod" command, which parallels "mymethod".

\* Added the "myvar" and "mytypevar" commands. "varname" is now

deprecated.

\* Added ::snit::macro.

\* Added the "component" type definition statement. This replaces

"variable" for declaring components explicitly, and has two nifty

options, "-public" and "-inherit".

\* Reimplemented the "delegate method" and "delegate option"

statements; among other things, they now have more descriptive error

messages.

\* Added the "using" clause to the "delegate method" statement. The

"using" clause allows the programmer to specify an arbitrary command

prefix into which the component and method names (among other

things) can be automatically substituted. It's now possible to

delegate a method just about any way you'd like.

\* Added ::snit::compile.

\* Added the "delegate typemethod" statement. It's similar to

"delegate method" and has the same syntax, but delegates typemethods

to commands whose names are stored in typevariables.

\* Added the "typecomponent" type definition statement. Parallel to

"component", "typecomponent" is used to declare targets for the new

"delegate typemethod" statement.

\* "delegate method" can now delegate methods to components or

typecomponents.

\* The option definition syntax has been extended; see snit.man. You

can now define methods to handle cget or configure of any option; as

a result, The "oncget" and "onconfigure" statements are now deprecated.

Existing "oncget" and "onconfigure" handlers continue to function as

expected, with one difference: they get a new implicit argument,

"\_option", which is the name of the option being set. If your

existing handlers use "\_option" as a variable name, they will need

to be changed.

\* In addition, the "option" statement also allows you to define a

validation method. If defined, it will be called before the value

is saved; its job is to validate the option value and call "error"

if there's a problem.

\* In addition, options can be defined to be "-readonly". A readonly

option's value can be set at creation time (i.e., in the type's

constructor) but not afterwards.

\* There's a new type definition statement called "pragma" that

allows you to control how Snit generates the type from the

definition. For example, you can disable all standard typemethods

(including "create"); this allows you to use snit::type to define

an ensemble command (like "string" or "file") using typevariables

and typemethods.

\* In the past, you could create an instance of a snit::type with the

same name as an existing command; for example, you could create an

instance called "::info" or "::set". This is no longer allowed, as

it can lead to errors that are hard to debug. You can recover the

old behavior using the "-canreplace" pragma.

\* In type and widget definitions, the "variable" and "typevariable"

statements can now initialize arrays as well as scalars.

\* Added new introspection commands "$type info typemethods",

"$self info methods", and "$self info typemethods".

\* Sundry other internal changes.

Changes in V0.94

--------------------------------------------------------------------

V0.94 was the development version in which most of the V0.95 changes

were implemented. The name was changed to V0.95 when the changes

were complete, so that the contents of V0.95 will be stable.

Changes in V0.93

--------------------------------------------------------------------

\* Enhancement: Added the snit::typemethod and snit::method commands;

these allow typemethods and methods to be defined (and redefined)

after the class already exists. See the Snit man page for

details.

\* Documentation fixes: a number of minor corrections were made to the

Snit man page and FAQ. Thanks to everyone who pointed them out,

especially David S. Cargo.

\* Bug fix: when using %AUTO% to create object names, the counter

will wrap around to 0 after it reaches (2^32 - 1), to prevent

integer overflow errors. (Credit Marty Backe)

\* Bug fix: in a normal Tcl proc, the command

variable ::my::namespace::var

makes variable "::my::namespace::var" available to the proc under the

local name "var". Snit redefines the "variable" command for use in

instance methods, and had lost this behavior. (Credit Jeff

Hobbs)

\* Bug fix: in some cases, the "info vars" instance method didn't

include the "options" instance variable in its output.

\* Fixed bug: in some cases the type command was created even if there

was an error defining the type. The type command is now cleaned

up in these cases. (Credit Andy Goth)

Changes in V0.92

--------------------------------------------------------------------

\* Bug fix: In type methods, constructors, and methods, the "errorCode"

of a thrown error was not propagated properly; no matter what it was

set to, it always emerged as "NONE".

Changes in V0.91

--------------------------------------------------------------------

\* Bug fix: On a system with both 0.9 and 0.81 installed,

"package require snit 0.9" would get snit 0.81. Here's why: to me

it was clear enough that 0.9 is later than 0.81, but to Tcl the

minor version number 9 is less than minor version number 81.

From now on, all pre-1.0 Snit version numbers will have two

digits.

\* Bug fix: If a method or typemethod had an argument list which was

broken onto multiple lines, the type definition would fail. It now

works as expected.

\* Added the "expose" statement; this allows you to expose an entire

component as part of your type's public interface. See the man page

and the Snit FAQ list for more information.

\* The "info" type and instance methods now take "string match"

patterns as appropriate.

Changes in V0.9

--------------------------------------------------------------------

For specific changes, please see the file ChangeLog in this directory.

Here are the highlights:

\* Snit widgets and widget adaptors now support the Tk option database.

\* It's possible set the hull type of a Snit widget to be either a

frame or a toplevel.

\* It's possible to explicitly set the widget class of a Snit widget.

\* It's possible to explicitly set the resource and class names for

all locally defined and explicitly delegated options.

\* Option and method names can be excluded from "delegate option \*" by

using the "except" clause, e.g.,

delegate option \* to hull except {-borderwidth -background}

\* Any Snit type or widget can define a "type constructor": a body of

code that's executed when the type is defined. The type constructor

is typically used to initialize array-valued type variables, and to

add values to the Tk option database.

\* Components should generally be created and installed using the new

"install" command.

\* snit::widgetadaptor hulls should generally be created and installed

using the new "installhull using" form of the "installhull" command.

See the Snit man page and FAQ list for more information on these new

features.

Changes in V0.81

--------------------------------------------------------------------

\* All documentation errors people e-mailed to me have been fixed.

\* Bug fix: weird type names. In Snit 0.8, type names like

"hyphenated-name" didn't work because the type name is used as a

namespace name, and Tcl won't parse "-" as part of a namespace name

unless you quote it somehow. Kudos to Michael Cleverly who both

noticed the problem and contributed the patch.

\* Bug fix: Tcl 8.4.2 incompatibility. There was a bug in Tcl 8.4.1

(and in earlier versions, likely) that if the Tcl command "catch"

evaluated a block that contained an explicit "return", "catch"

returned 0. The documentation evidently indicated that it should

return 2, and so this was fixed in Tcl 8.4.2. This broke a bit

of code in Snit.

Changes in V0.8

--------------------------------------------------------------------

\* Note that there are many incompatibilities between Snit V0.8 and

earlier versions; they are all included in this list.

\* Bug fix: In Snit 0.71 and Snit 0.72, if two instances of a

snit::type are created with the same name, the first instance's

private data is not destroyed. Hence, [$type info instances] will

report that the first instance still exists. This is now fixed.

\* Snit now requires Tcl 8.4, as it depends on the new command

tracing facility.

\* The snit::widgettype command, which was previously deprecated, has

now been deleted.

\* The snit::widget command has been renamed snit::widgetadaptor; its

usage is unchanged, except that the idiom "component hull is ..."

is no longer used to define the hull component. Instead, use the

"installhull" command:

constructor {args} {

installhull [label $win ...]

$self configurelist $args

}

\* The "component" command is now obsolete, and has been removed.

Instead, the "delegate" command implicitly defines an instance

variable for the named component; the constructor should assign an

object name to that instance variable. For example, whereas you

used to write this:

snit::type dog {

delegate method wag to tail

constructor {args} {

component tail is [tail $self.tail -partof self]

}

method gettail {} {

return [component tail]

}

}

you now write this:

snit::type dog {

delegate method wag to tail

constructor {args} {

set tail [tail $self.tail -partof self]

}

method gettail {} {

return $tail

}

}

\* There is a new snit::widget command; unlike snit::widgetadaptor,

snit::widget automatically creates a Tk frame widget as the hull

widget; the constructor doesn't need to create and set a hull component.

\* Snit objects may now be renamed without breaking; many of the

specific changes which follow are related to this. However,

there are some new practices for type authors to follow if they wish

to write renameable types and widgets. In particular,

\* In an instance method, $self will always contain the object's

current name, so instance methods can go on calling other instance

methods using $self.

\* If the object is renamed, then $self's value will change. Therefore,

don't use $self for anything that will break if $self changes.

For example, don't pass a callback as "[list $self methodname]".

\* If the object passes "[list $self methodname arg1 arg2]" as a callback,

the callback will fail when the object is renamed. Instead, the

object should pass "[mymethod methodname arg1 arg2]". The [mymethod]

command returns the desired command as a list beginning with a

name for the object that never changes.

For example, in Snit V0.71 you might have used this code to call a

method when a Tk button is pushed:

.btn configure -command [list $self buttonpress]

This still works in V0.8--but the callback will break if your

instance is renamed. Here's the safe way to do it:

.btn configure -command [mymethod buttonpress]

\* Every object has a private namespace; the name of this namespace

is now available in method bodies, etc., as "$selfns". This value is

constant for the life the object. Use "$selfns" instead of "$self" if

you need a unique token to identify the object.

\* When a snit::widget's instance command is renamed, its Tk window

name remains the same--and is still extremely important.

Consequently, the Tk window name is now available in snit::widget

method bodies, etc., as "$win". This value is constant for the

life of the object. When creating child windows, it's best to

use "$win.child" rather than "$self.child" as the name of the

child window.

\* The names "selfns" and "win" may no longer be used as explicit argument

names for typemethods, methods, constructors, or onconfigure

handlers.

\* procs defined in a Snit type or widget definition used to be able to

reference instance variables if "$self" was passed to them

explicitly as the argument "self"; this is no longer the case.

\* procs defined in a Snit type or widget definition can now reference

instance variables if "$selfns" is passed to them explicitly as the

argument "selfns". However, this usage is deprecated.

\* All Snit type and widget instances can be destroyed by renaming the

instance command to "".

Changes in V0.72

--------------------------------------------------------------------

\* Updated the pkgIndex.tcl file to references snit 0.72 instead of

snit 0.7.

\* Fixed a bug in widget destruction that caused errors like

"can't rename "::hull1.f": command doesn't exist".

Changes in V0.71

--------------------------------------------------------------------

\* KNOWN BUG: The V0.7 documentation implies that a snit::widget can

serve as the hull of another snit::widget. Unfortunately, it

doesn't work. The fix for this turns out to be extremely

complicated, so I plan to fix it in Snit V0.8.

Note that a snit::widget can still be composed of other

snit::widgets; it's only a problem when the hull component in

particular is a snit::widget.

\* KNOWN BUG: If you rename a Snit type or instance command (i.e., using

Tcl's [rename] command) it will no longer work properly. This is

part of the reason for the previous bug, and should also be fixed in

Snit V0.8.

\* Enhancement: Snit now preserves the call stack (i.e., the

"errorInfo") when rethrowing errors thrown by Snit methods,

typemethods, and so forth. This should make debugging Snit types

and widgets much easier. In Snit V0.8, I hope to clean up the

call stack so that Snit internals are hidden.

\* Bug fix: Option default values were being processed incorrectly. In

particular, if the default value contained brackets, it was treated

as a command interpolation. For example,

option -regexp {[a-z]+}

yield the error that "a-z" isn't a known command. Credit to Keith

Waclena for finding this one.

\* Bug fix: the [$type info instances] command failed to find

instances that weren't defined in the global namespace, and found

some things that weren't instances. Credit to Keith Waclena for

finding this one as well.

\* Internal Change: the naming convention for instance namespaces

within the type namespace has changed. But then, your code

shouldn't have depended on that anyway.

\* Bug fix: snit::widget destruction was seriously broken if the hull

component was itself a megawidget (e.g., a BWidget).

Each layer of megawidget code needs its opportunity

to clean up properly, and that wasn't happening. In addition, the

snit::widget destruction code was bound as follows:

bind $widgetName <Destroy> {....}

which means that if the user of a Snit widget needs to bind to

<Destroy> on the widget name they've just wiped out Snit's

destructor. Consequently, Snit now creates a bindtag called

Snit<widgettype>

e.g.,

Snit::rotext

and binds its destroy handler to that. This bindtag is inserted in

the snit::widget's bindtags immediately after the widget name.

Destruction is always going to be somewhat tricky when multiple

levels of megawidgets are involved, as you need to make sure that

the destructors are called in inverse order of creation.

Changes in V0.7

----------------------------------------------------------------------

\* INCOMPATIBILITY: Snit constructor definitions can now have arbitrary

argument lists, as methods do. That is, the type's create method

expects the instance name followed by exactly the arguments defined

in the constructor's argument list:

snit::type dog {

variable data

constructor {breed color} {

set data(breed) $breed

set data(color) $color

}

}

dog spot labrador chocolate

To get the V0.6 behavior, use the argument "args". That is, the

default constructor would be defined in this way:

snit::type dog {

constructor {args} {

$self configurelist $args

}

}

\* Added a "$type destroy" type method. It destroys all instances of

the type properly (if possible) then deletes the type's namespace

and type command.

Changes in V0.6

-----------------------------------------------------------------

\* Minor corrections to the man page.

\* The command snit::widgettype is deprecated, in favor of

snit::widget.

\* The variable "type" is now automatically defined in all methods,

constructors, destructors, typemethods, onconfigure handlers, and

oncget handlers. Thus, a method can call type methods as "$type

methodname".

\* The new standard instance method "info" is used for introspection on

type and widget instances:

$object info type

Returns the object's type.

$object info vars

Returns a list of the object's instance variables (excluding Snit

internal variables). The names are fully qualified.

$object info typevars

Returns a list of the object's type's type variables (excluding

Snit internal variables). The names are fully qualified.

$object info options

Returns a list of the object's option names. This always

includes local options and explicitly delegated options. If

unknown options are delegated as well, and if the component to

which they are delegated responds to "$object configure" like Tk

widgets do, then the result will include all possible unknown

options which could be delegated to the component.

Note that the return value might be different for different

instances of the same type, if component object types can vary

from one instance to another.

\* The new standard typemethod "info" is used for introspection on

types:

$type info typevars

Returns a list of the type's type variables (excluding Snit

internal variables).

$type info instances

Returns a list of the instances of the type. For non-widget

types, each instance will be the fully-qualified instance command

name; for widget types, each instance will be a widget name.

\* Bug fixed: great confusion resulted if the hull component of a

snit::widgettype was another snit::widgettype. Snit takes over the

hull widget's Tk widget command by renaming it to a known name, and

putting its own command in its place. The code made no allowance

for the fact that this might happen more than once; the second time,

the original Tk widget command would be lost. Snit now ensures that

the renamed widget command is given a unique name.

\* Previously, instance methods could call typemethods by name, as

though they were normal procs. The downside to this was that

if a typemethod name was the same as a standard Tcl command, the

typemethod shadowed the standard command in all of the object's

code. This is extremely annoying should you wish to define a

typemethod called "set". Instance methods must now call typemethods

using the type's command, as in "$type methodname".

\* Typevariable declarations are no longer required in

typemethods, methods, or procs provided that the typevariables are defined

in the main type or widget definition.

\* Instance variable declarations are no longer required in methods provided

that the instance variables are defined in the main type or widget

declaration.

\* Instance variable declarations are no longer required in procs,

provided that the instance variables are defined in the main type or

widget declaration. Any proc that includes "self" in its argument

list will pick up all such instance variables automatically.

\* The "configure" method now returns output consistent with Tk's when

called with 0 or 1 arguments, i.e., it returns information about one

or all options. For options defined by Snit objects, the "dbname"

and "classname" returned in the output will be {}. "configure" does

its best to do the right thing in the face of delegation.

\* If the string "%AUTO%" appears in the "name" argument to "$type create"

or "$widgettype create", it will be replaced with a string that

looks like "$type$n", where "$type" is the type name and "$n" is

a counter that's incremented each time a

widget of this type is created. This allows the caller to create

effectively anonymous instances:

widget mylabel {...}

set w [mylabel .pane.toolbar.%AUTO% ...]

$w configure -text "Some text"

\* The "create" typemethod is now optional for ordinary types so long

as the desired instance name is different than any typemethod name

for that type. Thus, the following code creates two dogs, ::spot

and ::fido.

type dog {...}

dog create spot

dog fido

If there's a conflict between the instance name and a typemethod,

either use "create" explicitly, or fully qualify the instance name:

dog info -color black ;# Error; assumes "info" typemethod.

dog create info -color black ;# OK

dog ::info -color black ;# also OK

\* Bug fix: If any Snit method, typemethod, constructor, or onconfigure

handler defines an explicit argument called "type" or "self", the type

definition now throws an error, preventing confusing runtime

behavior.

\* Bug fix: If a Snit type or widget definition attempts to define a

method or option locally and also delegate it to a component, the

type definition now throws an error, preventing confusing runtime

behavior.

\* Bug(?) Fix: Previously, the "$self" command couldn't be used in

snit::widget constructors until after the hull component was

defined. It is now possible to use the "$self" command to call

instance methods at any point in the snit::widget's

constructor--always bearing in mind that it's an error to configure

delegated options or are call delegated methods before creating the

component to which they are delegated.

Changes in V0.5

------------------------------------------------------------------

\* Updated the test suite so that Tk-related tests are only run if

Tk is available. Credit Jose Nazario for pointing out the problem.

\* For snit::widgettypes, the "create" keyword is now optional when

creating a new instance. That is, either of the following will

work:

::snit::widgettype mylabel { }

mylabel create .lab1 -text "Using create typemethod"

mylabel .lab2 -text "Implied create typemethod"

This means that snit::widgettypes can be used identically to normal

Tk widgets. Credit goes to Colin McCormack for suggesting this.

\* Destruction code is now defined using the "destructor" keyword

instead of by defining a "destroy" method. If you've been

defining the "destroy" method, you need to replace it with

"destructor" immediately. See the man page for the syntax.

\* widgettype destruction is now handled properly (it was buggy).

Use the Tk command "destroy" to destroy instances of a widgettype;

the "destroy" method isn't automatically defined for widgettypes as

it is for normal types, and has no special significance even if it

is defined.

\* Added the "from" command to aid in parsing out specific option

values in constructors.

Changes in V0.4

------------------------------------------------------------------

\* Added the "codename" command, to qualify type method and private

proc names.

\* Changed the internal implementation of Snit types and widget types

to prevent an obscure kind of error and to make it easier to pass

private procs as callback commands to other objects. Credit to Rolf

Ade for discovering the hole.

Changes in V0.3

------------------------------------------------------------------

\* First public release.