Sharpen : A Static Analysis Tool For Vignette's "TCL"

Brian Passingham
Passingham Software Ltd
327 Oxford Road
Macclesfield
Cheshire SK11 8JZ
England

E-mail: brian@passisoft.com

http://www.passisoft.com/sharpen.html

ABSTRACT

Early versions of the Vignette™ content management system (up to version 6) make use of an extension of the Tcl programming language, which we shall refer to as "TCL" to distinguish it from the mainstream Tcl language. We briefly discuss how various features of "TCL" can encourage a particular set of "Frequently Made Mistakes", leading to high development costs. Unfortunately, "TCL" is not understood by the best and most popular Tcl IDEs, and so these cannot be used to address these issues. We present a new tool, Sharpen, which can cope with the "TCL" extensions, and discuss how it can be used to reduce the maintenance costs of those Vignette legacy sites still using the "TCL"-based product. We also outline some possible future applications of the technology.

Introduction - Vignette and "TCL"

Early versions of the Vignette™ content management system (up to version 6) make use of an extension of the Tcl programming language, which we shall refer to as "TCL" to distinguish it from the mainstream Tcl language. The behaviour described here is that of the most commonly encountered configuration post V/5, in which interpreter re-use mode is enabled. We also assume that the original StoryServer behaviour with respect to EVAL's use of backslashes and SET's use of EVAL is present. V/6 provides options for more sensible functionality in these areas, but the costs and risks of updating a codebase to make use of these options seem to deter companies from making use of them, except, obviously, on new installations.

For a more complete description of the architecture of StoryServer, as the Vignette/TCL product was initially named, refer to the Vignette documentation. For our purposes here all we need to know is that a pool of page generator processes, each containing a Tcl interpreter, are used to deliver dynamic web pages by evaluating "TCL" templates. Template code is held in a database, and propagated to public-facing ("live") and private ("development") web servers. Typically, a content management application is run on the "development" web servers and used to publish data through to the live servers. Libraries of code may be included in templates through either the INCLUDE command (for TCL code necessitating the use of EVAL) or the SOURCE command (for normal Tcl code). In interpreter re-use mode, the SOURCE ... PERSIST command may be used to load code once for the duration of a page generator's existence.

Each "TCL" template consists of text with Tcl commands embedded using []s. So far, so *subst*-like. However, Vignette wished to make scripted HTML generation as easy as possible, and so also introduced some control flow constructs of their own, aimed at the generation of text. A Vignette-specific mechanism for long comments of the form [# ...] is also supported. Thus a simple template generating a barely formatted table of some selected fiction might look like this:

```
 [# Template: Simple Template
    Path : /simpleminded/library
    ... further identification details ...]

<table>
[SEARCH TABLE books INTO bookDetails SQL "select * from  library"
 FOREACH book IN [SHOW books] {
      [IF {[FIELD $book category] == "novel" && [BookSatisfiesUser $book]} {
           [# Generate a table row for each novel the user "likes", according to their criteria]
            <tr><td>[FIELD $book author]</td>
                <td>[FIELD $book title]</td>
                <td>[FIELD $book date]</td></tr>
      }]
}]
</table>
```

Note that the bodies of the FOREACH and IF statements are treated in the same way as the template itself – they are passed to the Vignette EVAL command.

Some Frequently Made Mistakes

In some ways the most important type of mistake made with StoryServer is the common-or-garden Tcl coding error.

Web applications are all-too-commonly tested almost entirely by a black-box method. Path coverage tends, therefore, to be weaker than one would wish. In the context of a language which is parsed and compiled on demand this can lead to embarrassingly basic errors finding their way through to live sites. For example, I have seen all-too-many page generation failures which have causes as simple as this :

```
[IF {[diceThrow] == 6} {
     <p>[string strange [SHOW text] 0 20]</p>
 }]
```

The frequent need to refer to characters needing a backslash is also often a cause of syntax errors, since code within N levels of EVAL requires 2 to the power N backslashes to be used! Sixteen backslashes look much like fifteen to the naked eye...

There are also subtler, Vignette-specific issues to contend with.

Consider the case where a developer omits a call to SOURCE a frequently called library. Statistically, it is very likely that any particular interpreter in the development environment will already have loaded the library. The code will only fail when the template is the first one called from a page generation process. Once the failure has occurred, of course, it is more likely to occur again, since a new page generator instance will be restarted...

There is also a similar, but thankfully rarer, problem in which code in a re-used library will only work on its first call, since its use invalidates its own pre-condition.

Mismatches in data scoping and control flow behaviour between the Vignette commands and "normal" Tcl can also cause problems – a full description of these would divert this paper unduly. Suffice to say that best practice at StoryServer sites tends to frown on the excessive usage of SET and SHOW.

If we're lucky (read disciplined!), the ease of making mistakes just leads to additional costs and delays when the problem is detected in regression testing. If we're unlucky, parts of the live web sites either fail completely, or misbehave in subtle ways. If we're really unlucky, there are enough problems to start destabilizing the system to a point where the delivery of any web pages at all becomes problematic.

For this reason alone, even a basic parser and usage checker for StoryServer can lead to important quality improvements.

Unfortunately, the main Tcl IDEs such as ActiveState's TclDev and T-IDE (apologies to any I've missed) provide no support for StoryServer. The issue of accessing code in a database can easily be resolved, of course. The issue is that of parsing TCL. The precise syntax of Vignette's [#...] comments is undocumented, and the excess backslashes of EVALed code defeat Tcl's own parser.

Implementation of Sharpen

For this reason we have implemented a parser package for TCL, ::**sharpen**.

We had initially hoped to build on Tcl's internal parser, as exposed by a Tcl interface in TclPro. This is clearly the best way of guaranteeing a correct parse. However, experiments in this direction proved the strategy not to be viable. The extra backslashes present (due to EVAL's backslash escaping) in most TCL code cause real difficulties. Working with a "backslash eliminated" version of the text, along with a map back to the original text, we made some headway, but abandoned the approach after considering the memory overheads (StoryServer codebases can be *big,* since multiple sites are often maintained in a single environment).

As a result our approach is broadly similar to that of the incomplete '**parsetcl**', though our representation of the parse tree differs considerably. We use an associative array to represent the parse tree, mainly since our code needs to run in an 8.2 context when tightly integrated with Vignette. We use a callback mechanism to reparse calls of procs which themselves have a significant syntactic component. We were tempted to drive this mechanism by a file of rules, as **Nagelfar** does, but decided that this might close the architecture - we intend to extend **::sharpen** to perform parses of several sublanguages.

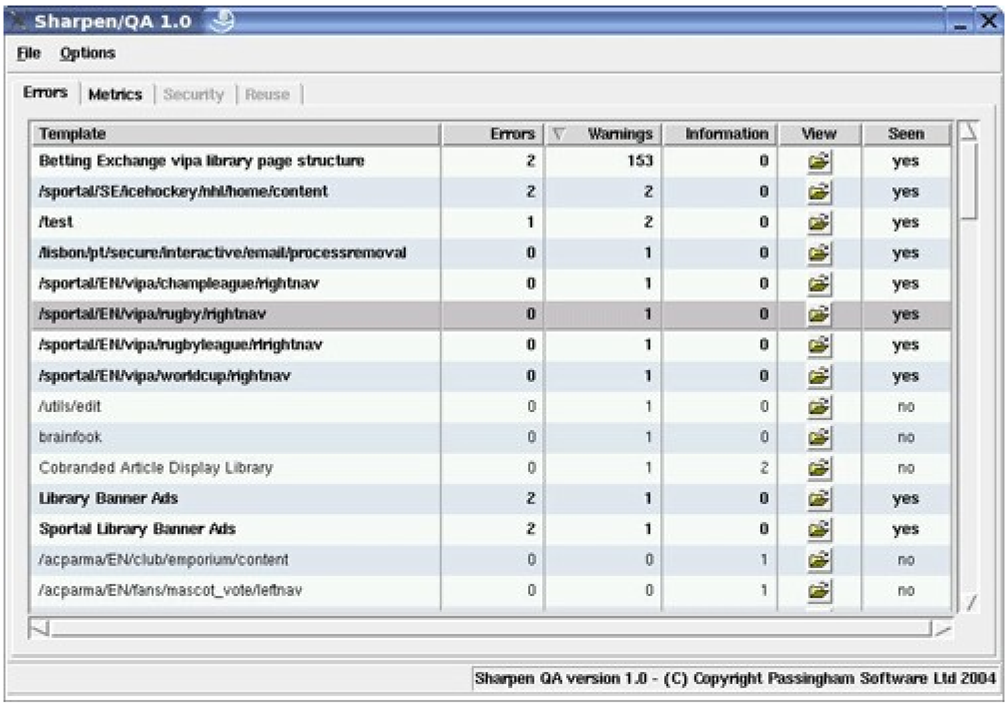
We also implemented a Vignette package parsing package, ::**vpkg**, to cope with the serialized data files produced by the *transferproject* utility used to move code and its meta-deta between environments. (It seems likely here that we have duplicated the efforts of the TrapEd product, which provides an editor for such packages.)

Current Capabilities of Sharpen

The current stable release of the Sharpen product consists of versions of the ::**sharpen** and ::**vpkg** packages, along with an example tool, **Sharpen/QA,** which uses these to provide basic QA facilities for Vignette TCL and Tcl templates, either individually, *en masse*, or from a package.

Sharpen/QA is a configurable tool for identifying quality assurance issues in Vignette TCL code, with support for reviewing individual templates, packages, or entire codebases. Reports from **Sharpen/QA** identify the *procs* defined and used by particular templates, allowing the cross-checking of software releases against particular environments. In addition to the basic capabilities of identifying syntax errors and usage problems, it is also possible to use Sharpen/QA to identify common problems such as attempts to release code that contains inappropriate calls (such as the use of ERROR_TRACE as a coarse-grained diagnostic). These static analysis features should reduce errors when undertaking releases. They are also of continual use during the development process. Integration with existing development tools may be possible, depending on the software used.

Sharpen/QA uses Csaba Nemethi's tablelist widget by default.

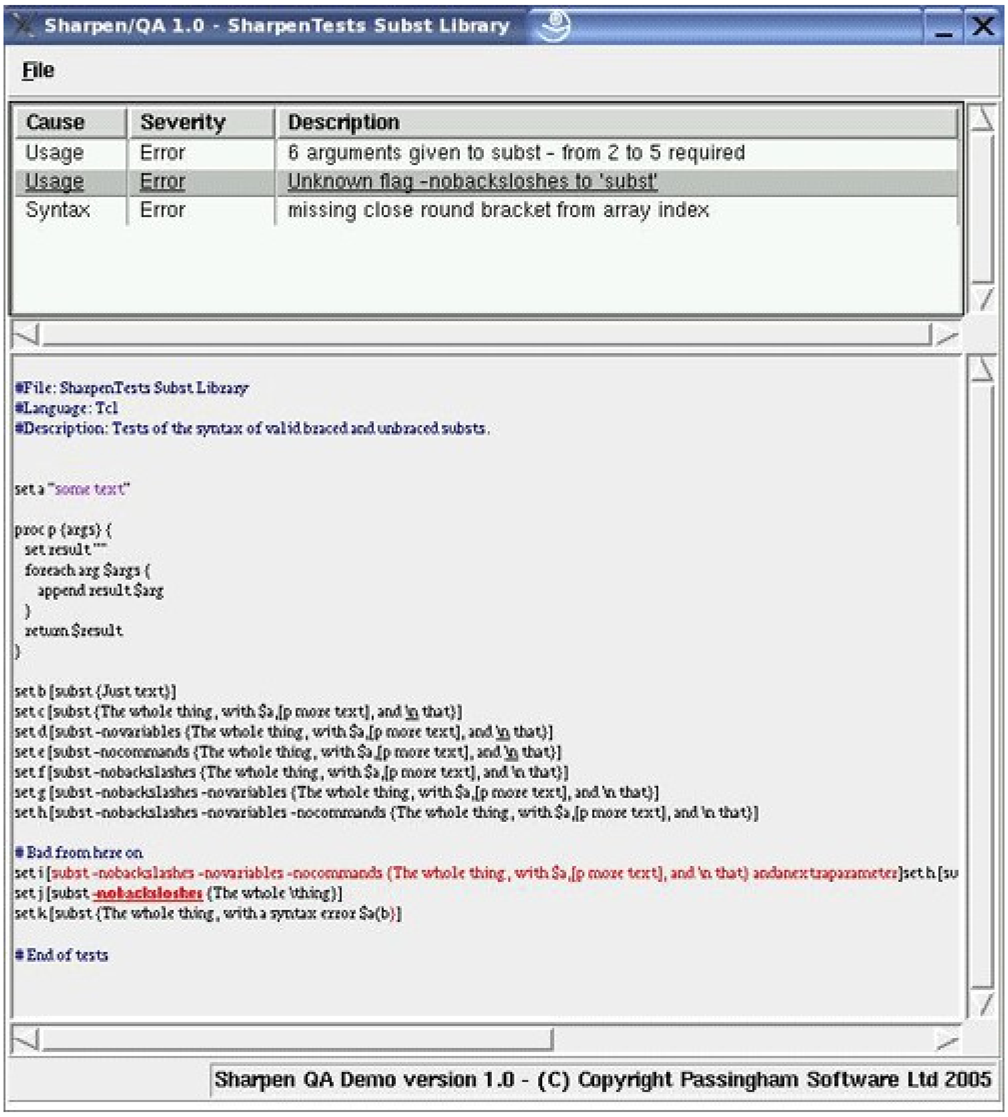


Template	Errors	Warnings	Information	View	Seen
Betting Exchange vipa library page structure	2	153	0		yes
/sportal/SE/icehockey/nhl/home/content	2	2	0		yes
/test	1	2	0		yes
/lisbon/pt/secure/interactive/email/processremoval	0	1	0		yes
/sportal/EN/vipa/champleague/rightnav	0	1	0		yes
/sportal/EN/vipa/rugby/rightnav	0	1	0		yes
/sportal/EN/vipa/rugbyleague/rrightnav	0	1	0		yes
/sportal/EN/vipa/worldcup/rightnav	0	1	0		yes
/utils/edit	0	1	0		no
brainfook	0	1	0		no
Cobranded Article Display Library	0	1	2		no
Library Banner Ads	2	1	0		yes
Sportal Library Banner Ads	2	1	0		yes
/acparma/EN/club/emporium/content	0	0	1		no
/acparma/EN/fans/mascot_vote/leftnav	0	0	1		no

Sharpen QA version 1.0 - (C) Copyright Passingham Software Ltd 2004

The above screenshot shows the Error frame following analysis of a codebase, with templates shown along with the number of categorised issues found within them. When applied to the now-defunct Sportal codebase (which supported the main Sportal portal as well as the websites of numerous top-ranked European football teams), Sharpen's QA tool uncovered numerous templates with serious problems that would have led to page generation failures, along with numerous instances of inefficient or risky coding practices. Experience to date suggests that this pattern is likely to be repeated at many Vignette TCL installations.

The following screenshot shows the Sharpen/QA template viewer, built using a Tk text widget tagged by the parse tree and its annotations. Due to the particular tag settings in use, it shows lightly syntax-highlighted TCL code (normal Tcl code in this example). Several errors have been identified and highlighted in red.


Sharpen/QA 1.0 - SharpenTests Subst Library
File
Sharpen QA Demo version 1.0 - (C) Copyright Passingham Software Ltd 2005

Cause	Severity	Description
Usage	Error	6 arguments given to subst - from 2 to 5 required
Usage	Error	Unknown flag -nobacksloshes to 'subst'
Syntax	Error	missing close round bracket from array index

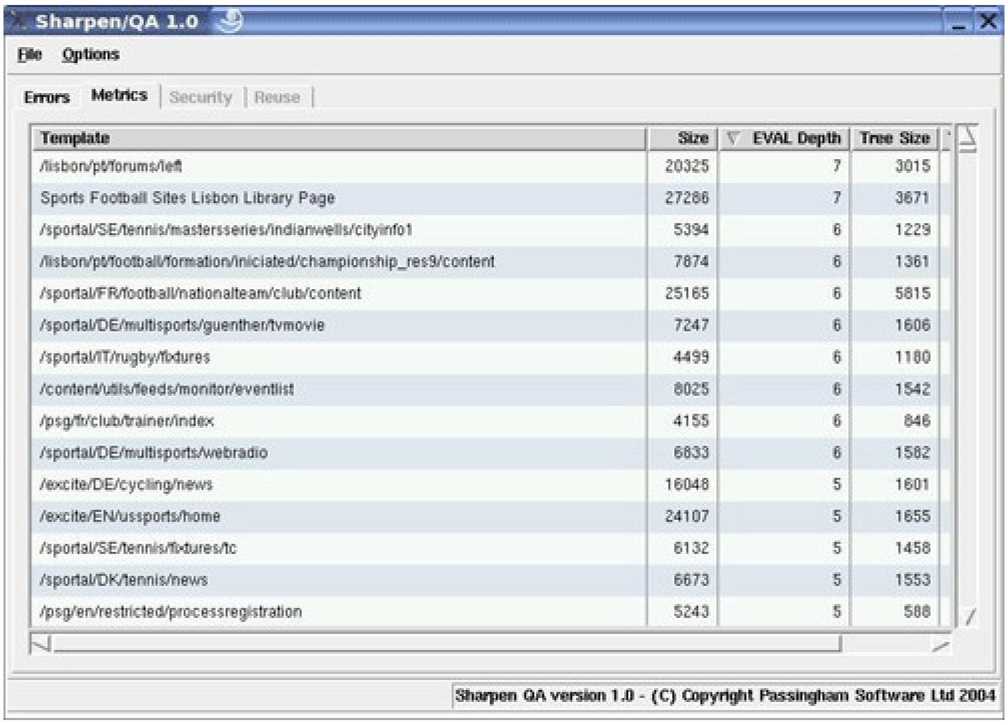
```
#File: SharpenTests Subst Library
#Language: Tcl
#Description: Tests of the syntax of valid braced and unbraced substs.


set a "some text"

proc p {args} {
  set result ""
  foreach arg $args {
    append result $arg
  }
  return $result
}

set b [subst {Just text}]
set c [subst {The whole thing, with $a,[p more text], and \n that}]
set d [subst -novariables {The whole thing, with $a,[p more text], and \n that}]
set e [subst -nocommands {The whole thing, with $a,[p more text], and \n that}]
set f [subst -nobackslashes {The whole thing, with $a,[p more text], and \n that}]
set g [subst -nobackslashes -novariables {The whole thing, with $a,[p more text], and \n that}]
set h [subst -nobackslashes -novariables -nocommands {The whole thing, with $a,[p more text], and \n that}]

# Bad from here on
set i [subst -nobackslashes -novariables -nocommands {The whole thing, with $a,[p more text], and \n that} andanextraparameter]set h [su
set j [subst -nobacksloshes {The whole \thing}]
set k [subst {The whole thing, with a syntax error $a(b)}]

# End of tests
```

The next screenshot shows the statistics of the Metrics frame, which can be used to readily identify a site's most complex templates (since these are likely to require the most testing/development effort). The statistics shown here are:

- File size - the simplest measure of a template's complexity, which can be misleading. There is nothing inherently complex about a flat "terms and conditions" template, for example.

- Tree size - a more accurate measure of the template's complexity, recording the size of the parse tree (as well as some overhead caused by repeated reparsing). There is some evidence from Beizer that claims a significant correlation between this type of figure and the expected number of errors caused by the code.

- EVAL depth - a Vignette-specific metric, recording the extent to which commands like EVAL, IF, and FOREACH have been nested. Excessive nesting can lead to serious degradation in performance - we would recommend the use of normal Tcl control structures instead for all but the simplest of cases.



Template	Size	EVAL Depth	Tree Size
/lisbon/pt/forums/left	20325	7	3015
Sports Football Sites Lisbon Library Page	27286	7	3671
/sportal/SE/tennis/mastersseries/indianwells/cityinfo1	5394	6	1229
/lisbon/pt/football/formation/iniciated/championship_res9/content	7874	6	1361
/sportal/FR/football/nationalteam/club/content	25165	6	5815
/sportal/DE/multisports/guenther/tvmovie	7247	6	1606
/sportal/IT/rugby/fixtures	4499	6	1180
/content/utils/feeds/monitor/eventlist	8025	6	1542
/psg/fr/club/trainer/index	4155	6	846
/sportal/DE/multisports/webradio	6833	6	1582
/excite/DE/cycling/news	16048	5	1601
/excite/EN/ussports/home	24107	5	1655
/sportal/SE/tennis/fixtures/tc	6132	5	1458
/sportal/DK/tennis/news	6673	5	1553
/psg/en/restricted/processregistration	5243	5	588

Sharpen QA version 1.0 - (C) Copyright Passingham Software Ltd 2004

Future Work

The detailed parse trees obtained through **::sharpen** and **::vpkg** can be used as the basis of more sophisticated tools – **Sharpen/QA** is a fairly straightforward application designed to test the technology and provide immediate benefits to Vignette/TCL developers.

One key area for future work will be extending the tool's abilities in the area of data flow analysis. There are of course, theoretical obstacles here – it is always possible for a Tcl developer to produce an example of dynamic code that will defeat a particular analytical technique – but even a naïve data flow analysis will work on the majority of StoryServer code and be capable of detecting common errors, such as code branches in which particular variables are used before they are set.

We are considering supporting further metrics. In particular, interest has been shown in some form of McCabe's cyclomatic complexity metric. We believe that the conventional definition of this will need to be supplemented by a further metric indicating the presence of non-imperative coding structures. A preliminary search of the literature has failed to find anything immediately suitable.

Operations to transform code whilst preserving its meaning are definitely possible, and offer great potential.

Particularly useful operations might be:

1. To eliminate EVAL-based control structures such as IF and FOREACH in favour of a byte-compilable procs which use normal Tcl control structures and *append* to build up a result string. This can yield significant performance improvements.

2. Excess backslash elimination to allow code to be migrated to a V/6 installation using the "sane backslashing" configuration. Such code is undoubtedly easier to read, and will be easier to develop, simply by eliminating the common confusion as to how many backslashes are required at a particular point.

3. Code instrumentation, allowing the construction of a StoryServer debugger and/or a code coverage tool.

More ambitiously, a full-blown TCL refactoring editor is a tempting prospect, though, given the limited time-frame within which Vignette will continue to support their TCL-based products, we are unlikely to embark on this (for this purpose at least).

We intend to demonstrate some proof-of-concept implementations in some of these areas at the meeting, and will publish these on http://www.passisoft.com after the meeting.