# Managing Tcl's Namespaces Collaboratively

Don Libes
*National Institute of Standards and Technology*
*Gaithersburg, MD 20899*
*libes@nist.gov*

## Abstract

The NIST Identifier Collaboration Service (NICS) is a proposed service to encourage collaboration among researchers and developers when choosing identifiers, far in advance of when it might ordinarily occur. This would support and enhance standards development activities, and development and communications in a variety of fields from software development to system administration. Implementation of this system would provide immediate and significant time and cost-savings to many technology administrators, researchers, developers, and implementors world-wide.

This paper describes the benefits of NICS to the Tcl community. This paper also briefly describes the implementation of NICS which is Tcl-based internally.

Keywords: collaborative namespace management, data element registry, NICS, Tcl

## What's an identifier – So what's the problem?

An identifier identifies things. For example, in a programming language, variables are identifiers. You distinguish one variable from another because they have different names. In HTML, tags are identifiers. For example, h1, h2, h3 and so on, are all identifiers. System calls, library names, port numbers are all identifiers.

So what's the problem? The problem is that choosing the right identifier is tricky. I don't mean choosing from among existing identifiers. That's easy. Just read the manual – or the standard. No, the hard part is coming up with new identifiers. It sounds easy but it's hard.

## Tcl's Namespace Disaster

Consider Tcl. Tcl provides no scoping of commands [Ousterhout]. Thus, it is important to choose command names which are unique. This is particularly problematic when defining commands that are meant to work with anyone else's command extensions. Tcl commands are generally short English words or abbreviations. Despite the availability of namespace management extensions such as incr Tcl, command name collisions are common [McLennan]. Ad hoc solutions such as prefixing all commands with a string unique to the extension reduce the problem but do not solve it.

People may suggest that the promised namespace support may rid Tcl of collisions but this isn't quite what will happen. In fact, the namespaces themselves will then need to be managed. This situation of a shared namespace is not unique to Tcl but is common to any rapidly evolving system undergoing development by parties that have no direct means of or desire to collaborate frequently.

NICS provides a general-purpose registry for identifiers. It is designed specifically to be adaptable to different domains. By registering command names and prefixes with this service, it is possible to avoid the types of conflicts mentioned earlier. The remainder of the paper will describe NICS and how it can benefit Tcl with a very low cost to the Tcl development community. Keep in mind that Tcl is only one such application for NICS.

## More about NICS

NICS is a web-based collaboration service specifically for identifiers. It provides features that have come to be expected for any web service: world-wide availability and instantaneous access. The service is accessible to the public, yet all information is authenticated – all information is browseable, but all additions or modifications require authentication appropriate for the information. Registration and authentication are totally automated and immediate – from initial contact to full use. All use is free.

NICS is not merely an online registry. Rather, NICS has appropriate and innovative support specifically for identifier collaboration.

- Users may register identifiers well before their use, thereby allowing very early declarations of intent and feedback on ideas, even before initial prototypes or draft standards.
- Identifier descriptions must include their status with respect to use and standards. Placeholders identifiers, deprecated identifiers, and other non-official or de facto identifiers are encouraged if they clarify knowledge needed by other users.
- Users may register classes of identifiers.
- Users may temporarily register multiple identifiers for the same purpose. For example, inter-

nal disputes within standards teams on identifiers should not prevent dissemination and public comment on the possible choices being considered.
- Users may privately or publicly comment on other identifiers. For example, users can suggest better choices or identify potential conflicts. Public comments avoid other multiple users redundantly contacting the original user.
- Information is available instantaneously. There is no delay between the time identifiers and comments are added and when they can be read by other users.
- Identifiers are allowed to conflict. Temporary disagreements show where attention of research should be focused. It is also possible that conflicting identifiers may merely reflect the real world.
- If requested, conflicts and comments are immediately sent to the identifier owner so that they can respond promptly.

## Further application to Tcl

The obvious use of NICS for Tcl is to choosing command names. Extension writers could register their extension's command names. Then other extension writers could avoid choosing names that are already in use or are planned for the future. The alternative is to install all other extensions (or read about them) and look for conflicts. Obviously this is not feasible. And it is impossible to learn about future intents this way. In comparison, NICS is trivial to use.

The Tcl development team has preannounced support for namespaces. Although the details are not yet available, it is likely that this work will directly address some of the problems having to do with command name collisions.

However even assuming the command name collision problem disappeared, several other sources of collisions would remain.

- Namespace names – Namespaces themselves will of course have names. These names themselves can have collisions. Although this should be much less of a problem than command names, it is still a problem since extension authors must often pick these names in ignorance of other extensions.

- Package and App_Init names – Collisions occur frequently in this domain. Not surprisingly, many collisions occur because many packages tackle the same area. For example, the Tcl FAQ lists packages from six different authors to generate random numbers, three packages to handle the Berkeley DBlibrary and five packages to support objects. There are literally hundreds of conflicts like this. And there are also conflicts simply because of a poor choice of names, such as names that are too short to be unique or meaningful. In a description of packages, Ousterhout acknowledges the problem and goes on to advise "Choose prefixes that are relatively short (3 to 6 characters)" [Ousterhout]. It is not surprising that people have sometimes reacted by going to the other extreme – choosing lengthy (and sometimes equally unmeaningful) names in an attempt to avoid collisions.

- TCL_XXX values – The completion codes returned by Tcl commands share an unmanaged namespace. There are five predefined codes (TCL_OK, TCL_BREAK, TCL_CONTINUE, etc.). Anything else is application defined. There is no obvious way for extensions to avoid collisions with one another since these numbers are not easily obtained or manipulated. In my own experience with Expect [Libes95], I took the defensive strategy of using relatively large numbers, far, far away from Tcl's choices. This strategy was weak but there does not appear to be any better alternative.

  This is an example of a namespace that could in theory be managed automatically by Tcl. It's trickier than it first appears, since it must account for completion codes explicitly specified in scripts or passed between interpreters in different processes with potentially different completion mappings. Until such a manager is developed, NICS provides a simple solution for avoiding collisions.

- Math function names – Tcl permits the augmentation of function names for evaluation by expr. Like command collisions, function names that conflict simply replace earlier definitions.

- Library names – Everyone wants their library name to be meaningful and short. But after accounting for the "lib" prefix and versioned shared library suffixes such as "5.20.so", that leaves only four characters for portability to file systems with 14 character filenames, making the universe of library names small indeed. But even without these restrictions, most people choose short library names of no more than 6 or 7 characters.

Some of these examples are only part of a much larger problem. For instance, management of Tcl library names is just a subset of library names in general. Consider the system administrator faced with installing software "off the net". The administrator follows all the directions – configuring, compiling, installing – only to be hit at runtime with "ld.so: fatal: can't open file libQtvso.4, errno = 2" or something similar. At this point, a search of the Web reveals three different Qtv libraries!

Program function may require dealing with yet more collisions, apart from those intrinsic to Tcl. For example, programmers must choose port numbers statically and with trepidation. It's easy to tell what port numbers are in use on a host currently. And of course, you should avoid any ports listed in the IETF list of well-known ports [Reynolds]. However, you cannot account for future clashes. If someone else "claims" a port number and their program becomes "popular enough", you lose. Even if it doesn't become popular, anyone attempting to install both packages on their machine faces conflicts.

## Traditional approaches to public namespace management

There are common ways of avoiding collisions in shared namespaces:

- Managed database – The Perl Modules List is an example of collision avoidance by explicit human management [Bunce]. Hand-editted by a person, the list maintains all the well-supported Perl modules and provides placeholders for future module names. The advantages are obvious. Yet there are several drawbacks to this approach. The first is that it is expensive. And if the labor is volunteered, then the database becomes dependent on the good graces of that person and whatever they are willing to do and no more. Indeed, the Perl Modules List maintainer specifically refuses to maintain module location information, claiming understandably that it is too much work. Unfortunately, this makes the Modules List frustrating since a module can be listed at times and yet not appear to exist on CPAN (the official Perl archive) [CPAN].
- Published report or standard – I already mentioned the IETF publication and its drawbacks. By comparison, most traditional standards represent even less useful examples of timely information. Part of the problem is that any developing domain necessarily consists of a standard half and a non-standard half. It simply does not make sense to standardize the half of a domain that is in development. Yet practitioners in the field must be knowledgeable of both halves.

  Word of mouth, intuition, and dumb luck – This is the time-honored traditional approach to combining multiple Tcl extensions. Indeed, it is prevalent in many other domains. For example, linkers have no scoping mechanism. When linking together multiple libraries of C code, you have to cross your fingers and pray that subroutine names from different authors don't collide. If you've been given a library from a vendor without source, life can be very unpleasant.

Choosing identifiers is inherently collaborative. The whole point of using identifiers is so that everyone can tell what we are all talking about. If we were just talking to ourselves, we wouldn't need to identify our things. Yet we have no tools to help in this collaboration.

## What NICS looks like in use

NICS is a typical web application in the sense that it is CGI-based and uses HTML forms for interactive browsing and input. A file upload facility is available for large updates.

Initially the user selects a domain with which to browse. In figure 1, the user is selecting the domain "Tcl commands". Also shown are some of the other domains likely to be supported by NICS in the future.

Notice that domains do not have version numbers. There is no reason to distinguish, for example, between Tcl 7.4, 7.5, and 7.6 because it is possible to write scripts portable to all of them. Or to put it another way, if there are version-specific differences in the commands such that they could still be accounted for in the current Tcl, the differences should be documented among the commands themselves. Even older versions might not appear in NICS at all. For example, there is no value to listing Tcl 6.0-specific information in its own domain because no one is performing continued development of it or writing extensions for it. Note that historic or dep-

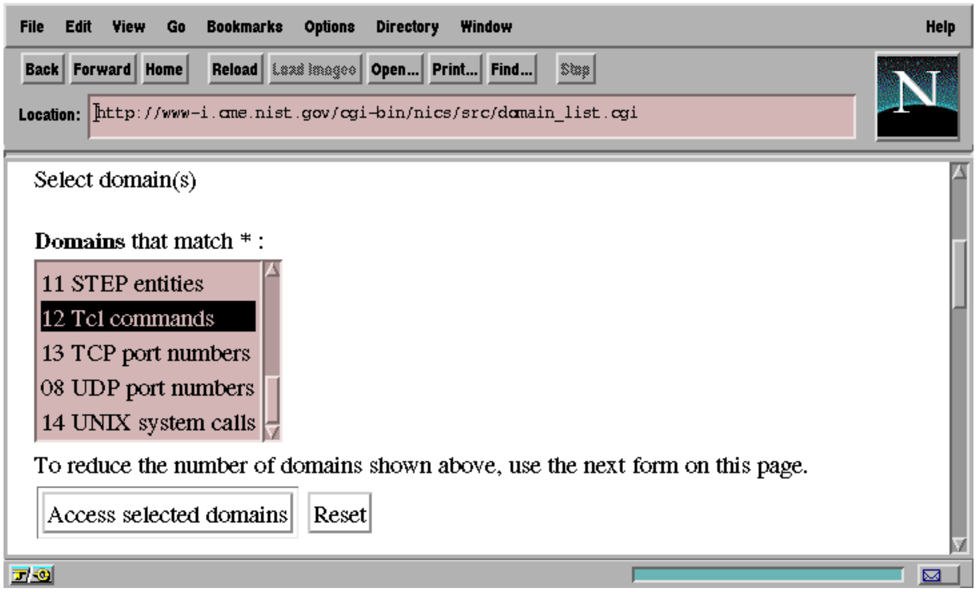


**Figure 1: Browsing and selecting domains from the many that NICS manages.**

recated commands can be listed (with such a proviso) if there is some useful reason to do so. For example, if an extension chose a name that was defunct but well-known in an earlier release, it might be useful to point this out.

Figure 2 shows a list of some of the identifiers in the domain – in this case Tcl command names. This list alone is useful for someone exploring the domain, perhaps while designing an extension. They can see the kinds of command names people choose, get a sense of the style that is expected, and most significantly avoid collisions with existing command names.

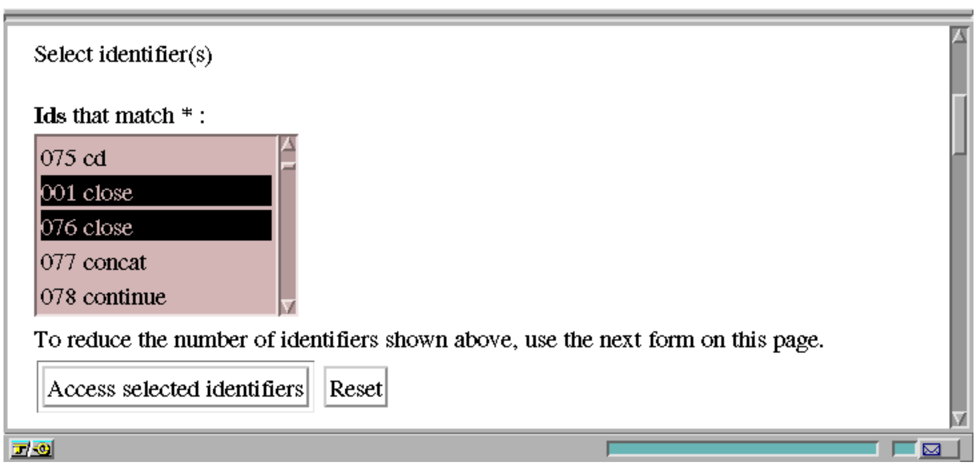


**Figure 2: Selecting two conflicting identifier names from the domain of Tcl command names.**

More information than just the name can be accessed by selecting particular identifiers. Here, the command name "close" has been selected – twice. Indeed, one definition is from Tcl and one is from Expect.

Although NICS has tools to discourage collisions, they are not prohibited. Collisions may come about because of any number of reasons – competing proposals, squabbling, or just ignorance. But the reality is, collisions exist in most domains undergoing rapid but decentralized development. Therefore, practitioners in the domain must be aware of these problems – or at least have some easy way of finding out.


**Figure 3: NICS shows the first close definition.**

Figure 3 shows detail on the definition for close provided by Tcl itself. Several attributes allow relatively arbitrary text such as the *brief description* and *references*. Since the service runs on the web, it is convenient to take advantage of hyperlinking. For example, the reference section here links to a man page on close that is provided courtesy of some other site. In fact, the textual fields allow arbitrary HTML allowing the possibility of .arbitrary formatting as well as the ability for inline images. In other domains, for example, we have made the descriptions appear to be directly from their standard.

Usage and standard status are also encouraged. Some of the choices for standard status include experimental, de facto, international. It is also possible to indicate "would be a mistake". This is intended to indicate an identifier that should not be standardized – perhaps because it is a placeholder such as unsupported0. This kind of identifier is more common than thought. Often, people don't give much thought to names or functions in a rush to get things working. They may recognize the limited future at the time but others may not and begin embedding such poor names or designs in stone when that was never their intent. Clearly, a name like unsupported0 is an excellent way to indicate the intent of the designer. However, few people are so bold and careful.

NICS can be told to contact the identifier owner if a collision occurs. This can encourage the two identifier owners to choose different (i.e., better) identifiers – or even better – to work together on the same one. At worst, the two owners will at least be made aware of the problem.

Owners may disable collision notification since such notification is not always appropriate. For example, a domain subset that has been standardized is not likely to be in the position to change.

Anyone can browse this information. However, the information is modifiable only by the identifier owner. In the example shown here, the identifier is a team: the Sun Labs Tcl/Tk Project. An owner may also be a person or any logical entity – as long as it has an email address. The authentication hinges on email reachability [Libes96a]. This technique provides moderate security (appropriate for NICS) with no administration cost to NIST and no special software required by users. Figure 4 shows the owner for the close command.[1]

Figure 5 shows the beginning of the second definition of close. It identifies itself as coming from the Expect extension. The only notable difference is that it declares an alias. Aliases are common in many domains. In the

1. Notice that the email authenticates who was actually responsible for adding the information.

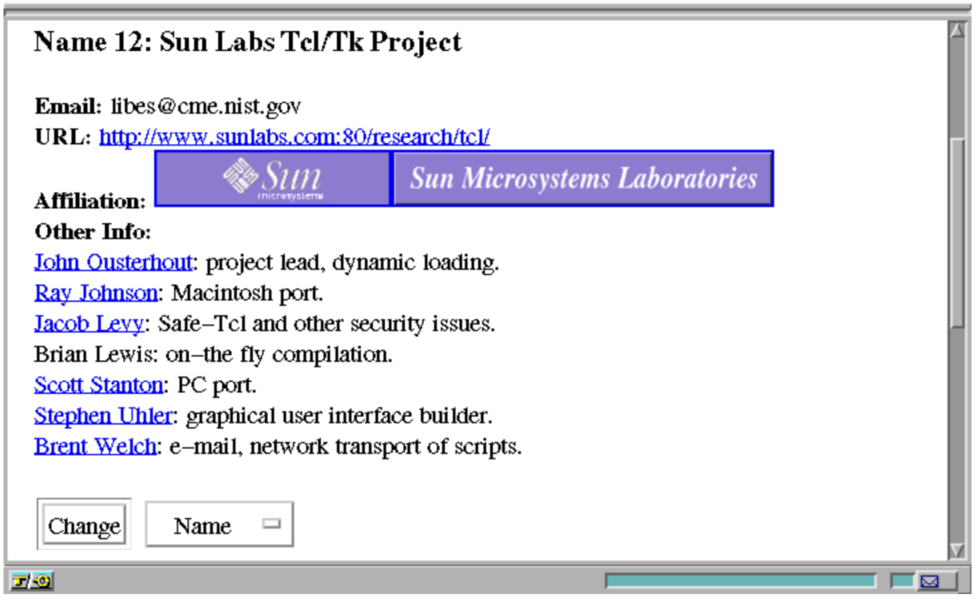


**Figure 4: NICS shows the owner of the first definition of the close command.**

case of Expect, all commands are also aliased with an exp_ prefix. Once informed of aliases, NICS allows queries as if they were full-fledged identifiers themselves but also remembers that they are defined by other identifiers.

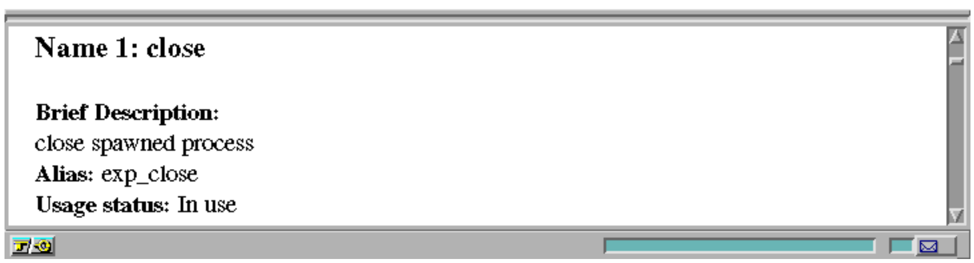


**Figure 5: Excerpt of an identifier showing a command name and alias.**

At the end of this particular identifier instance is a comment (figure 6). It has been placed by some other user who has observed that Expect's close appears to conflict with Tcl's.

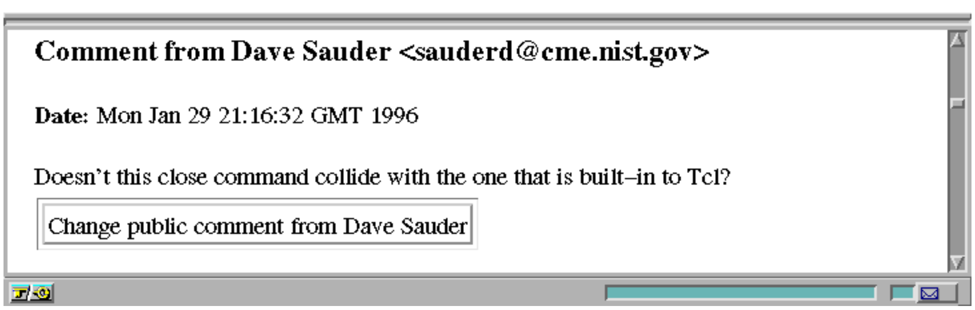


**Figure 6: A public comment attached to an identifier.**

Comments in practice tend to be more lengthy (and heated). Comments might also include suggestions for better names, references to both other extensions and people, historical work, future plans, etc. Indeed, it is not uncommon for some people to perform a surprisingly large amount of research in attempting to phrase the observation as carefully and intelligently as possible.

If comments were sent by private email, it is entirely possible that many other people would make the same observations leading to significant wasted effort. Attaching comments to identifiers avoids this. The identifier owner may respond to the comment privately or publicly but may not delete it. Only the comment owner may change or delete it.

Figure 7 shows a possible public response to the earlier comment. This is the type of information that could potentially stop many people from asking similar questions to the earlier one. The response appears after the original comment and can only be changed by the identifier owner.

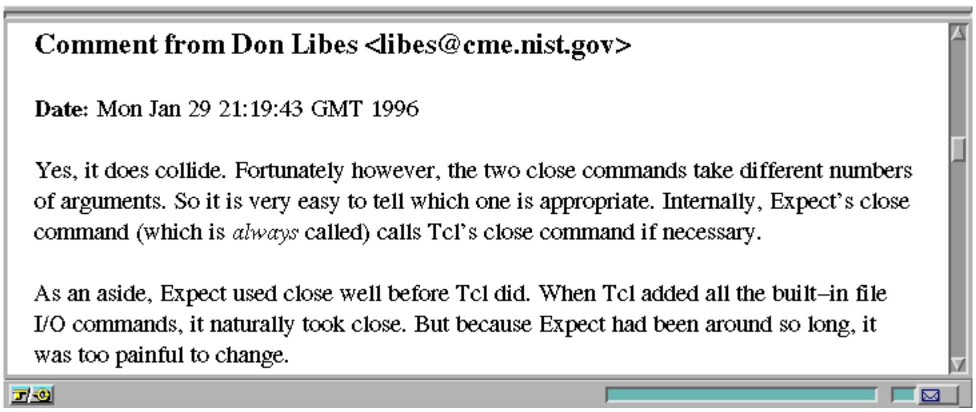


**Figure 7: Owner's response to a public comment.**

NICS performs notification of comments in a way similar to that of collisions. For example, identifier owners are notified (if they so request) of comments or changes to them.

## Other Features

The web page interface shown here allows identifiers to be added one at a time or en masse using a file upload facility. It is relatively easy, for example to automate population of the commands from an entire Tcl extension. For instance, the Expect commands were added by means of a 35-line script.

NICS has a number of other features. As an example, domains may be *moderated* meaning that the domain owner has control over who can add or comment on identifiers. However these other features are not particularly appropriate to the Tcl community and will not be covered here.

## Implementation Notes

The service is written entirely in Tcl. CGI support is provided by the Tcl-based CGI library, cgi.tcl (also entirely written in Tcl) [Libes96b]. Database service is provided by Oracle using OraTcl [Poindexter]. The service runs on a Sun Ultra with access to 55GB Raid storage running Netscape server software. NIST has an SMDS connection (34Mbps) to BBN Planet and a T3 connection (45Mbps) to MCI.[1]

With our current design, over the next three years, we envision on the order of 100,000 registered users, 500 domains, with an average of 10,000 identifiers per domain up to a peak of 1,000,000 identifiers for any one domain. We do not anticipate network bandwidth or space problems during this time. However, if demand outstrips our ability to provide space at a reasonable cost, we would look for and encourage other volunteers institutions to take over specific large domains.

1. Names of companies and products are provided in order to adequately specify procedures and equipment used. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

## Duration of the Service

Once development is complete, we envision the service running in an essentially automated mode. Usage and experience must be documented and additional disk space may be required from time to time, but the day-to-day operation will be totally automated. As long as the service is of significant value (i.e., that we claim it will be), we expect to be able to provide the service indefinitely, as the actual expenses (electricity, floor space, disk space, backups, maintenance) are a very small fraction of that of the NIST Information Processing Support Group.

## Applying NICS More Generally

Choosing identifiers collaboratively enables easier and faster development of standards and reuse of software. Implementation of this system would provide immediate and significant time and cost-savings to millions of software and standards developers world-wide.

Industry needs the benefits of this service but no single company wants to fund it – this project would benefit many while only the original company pays for it. In contrast, NIST, as a neutral site, is the ideal host for NICS.

Unlike the traditional standards activities performed by NIST, NICS is complementary. In particular, NIST performs research for standards and participates in the creation of standards. More and more, standards are growing increasingly complex. Many standards (STEP, POSIX, etc.) are taking 5 or even 10 years to complete. In part, this is because the technology and in some cases even the standards themselves are being created by many people without collaboration. Without giving up competitive advantages, these people want to collaborate yet they lack the mechanisms to do so in the area that NICS addresses.

Test audiences have identified a large number of domains to which NICS could be immediately applied. We intended on populating NICS with information from a number of these areas to catalyze usage. This is a time consuming task for the very reason that NICS is so useful – because this information is not easily accessible from one place or organized in any uniform way.

In addition to the technical work, crucial non-implementation work is also required. Specifically, various policies must be defined and implemented, and NICS must be thoroughly documented, both for immediate usability and for credibility as a self-sustaining project. Except for research issues, NICS is intended to run by itself as a production service in outlying years. This requires a significant amount of testing and focus on making sure that it is robust.

## Status

At the present time, NICS is running in a mode where it is restricted to a small number of domains: the Tcl domains and a few others. This represents an opportunity for the Tcl community – both to manage their namespaces using NICS, and to provide feedback to the usefulness of the concept. We are very interested in suggestions for improvements from the Tcl community which we view as a prototypical audience for the benefit of NICS.

With additional funding, we hope to open the service to other domains in the near future as well as complete other aspects of the service.

## Conclusion

NICS is a proposed service to encourage collaboration among researchers and developers when choosing identifiers, far in advance of when it might ordinarily occur. We believe NICS would be of direct benefit to the Tcl community in a variety of ways. While we expect that the one example of command-name conflicts may be addressed by advances in Tcl itself, other namespaces will remain a problem that can be addressed by NICS.

### Acknowledgments

Thanks to Mark Williamson, Tu Nguyen, and Jimmy Graham of the NIST Information Processing Support Group for providing web and Oracle service for NICS. Dan Nickchen implemented the first release of the Oracle integration from a prototype file-based version.

This project is financially supported in part by the NIST Advanced Technology Program. Earlier funding was received through a U.S. Department of Commerce Pioneer Grant and a grant from the NIST MEL Director's Reserve.

### References

[Bunce] Bunce, Tim, and Konig, Andreas, "The Perl 5 Module List", http://gd.tuwien.ac.at/languages/perl/CPAN/modules/00modlist.long.html.

[CPAN] "CPAN – Comprehensive Perl Archive Network", http://www.perl.com/cpan.

[Reynolds] J. Reynolds, J. Postel, “Assigned Numbers”, RFC 1700, USC – ISI, October 1994.

[Libes95] Libes, Don, “Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs”, O’Reilly and Associates, January 1995.

[Libes96a] Libes, D., “Authentication by Email Reception”, Proceedings of the Fifth System Administration, Networking, and Security Conference (SANS 96), Washington, DC, May 12-18, 1996.

[Libes96b] Libes, D., “Writing CGI Scripts in Tcl”, Proceedings of the Fourth Annual Tcl/Tk Workshop ‘96, Monterey, CA, July 10-13, 1996.

[McLennan] McLennan, Michael, “[incr tcl] – Object-Oriented Programming in Tcl, Proceedings of the 1993 Tcl/Tk Workshop, Berkeley, CA, June 10-11, 1993.

[Ousterhout] Ousterhout, John. K., “Tcl and the Tk Toolkit”, Addison-Wesley, 1994.

[Poindexter] Poindexter, Tom, “OraTcl”, Tcl/Tk Extensions, ed., Mark Harrison, O’Reilly & Associates, Inc., to appear.