An Object-Oriented Tcl/Tk Binding for Interpreted Control of the NIST EXPRESS Toolkit in the NIST STEP Application Protocol Development Environment

Don Libes
Stephen N. Clark

Manufacturing Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract

The National Institute of Standards and Technology (NIST) has built numerous software toolkits and applications for manipulating STEP and EXPRESS data. These toolkits are traditionally used as compiled libraries which are linked to other compiled modules.

This paper describes a binding allowing the toolkit interfaces to be called from interpreted scripts. This significantly reduces the time required to construct and compile new applications. An X11 extension allows the construction of graphic elements, providing easy creation and integration of existing applications into X graphic user interfaces. We describe how the combination of bindings has been used to construct a STEP Application Protocol Development Environment.

Keywords: Tcl; Tk; EXPRESS; Object Oriented; National PDES Testbed; PDES; STEP; APDE; Development Environment

Reprinted from Proceedings of the EXPRESS User Group Workshop (EUG '95), Grenoble, France, October 21-22, 1995.

Background

The PDES (Product Data Exchange using STEP) activity is the United States' effort in support of the Standard for the Exchange of Product Model Data (STEP). STEP is an evolving international standard for the interchange of product data between various vendors' CAD/CAM systems and other manufacturing-related software [1][2][3]. The National PDES Testbed has been established at the National Institute of Standards and Technology (NIST) to provide testing and validation facilities for the developing standard. The Testbed is funded by the Computer-aided Acquisition and Logistic Support (CALS) program of the Office of the Secretary of Defense.

As part of the testing effort, NIST is charged with providing software for manipulating STEP data. Provided in the form of tools and toolkits for building new tools, the software is research-oriented and evolving. This document is one of a set of reports ([4] - [14]) which describe various aspects of the software.

Typography and other Conventions

In this document, shell commands and output are set in **`Courier bold`**. EXPRESS source is set in Times Roman as is the rest of the text. Words or phrases being defined and placeholders that must be replaced by actual data are set in *Times Roman italic*. Optional elements are surrounded by brackets, [such as this phrase]. Occasionally fragments are quoted when they are very small or contain punctuation characters that might otherwise cause them to be confused with the surrounding text.

Introduction

The National Institute of Standards and Technology (NIST) has built numerous software toolkits and applications for manipulating STEP and EXPRESS data. The toolkits come as software libraries that may be compiled and linked into applications.

Traditionally, these applications have not had control languages. This means that an application concentrates on doing one thing. For example, the **`fedex`** application checks EXPRESS data for syntactic and semantic correctness. The **`shtolo`** application converts EXPRESS short forms to long forms. Other applications are similarly constrained.

Each of these applications generally does one thing well. While an application may also do other things, these ancillary functions are not its focus. Applying applications to other tasks may be difficult or impossible. Commonly, when an application is needed that does something slightly different from an already existing application, one of the following actions is taken:

- a new flag is added to the old program so that it can do both old and new tasks depending on the flag, or
- the old program is copied and customized to address the new application.

Both of these solutions require modification of the original source code, followed by recompilation, testing, and debugging. This is a drawback for development of new applications. In particular, developers should be able to reuse old applications much more easily.

To enable reuse, we have embedded the NIST EXPRESS Toolkit (hereafter called simply the "EXPRESS Toolkit") in an interpreted control language (Tcl). Using this we have begun development of tools written using the new extended language. This paper focuses on one such tool – an AP (Application Protocol) Development Environment (APDE) [15].

The APDE is a graphical integrated environment where a variety of tools must work together. Many tools required by the APDE are similar to existing ones. Yet the APDE has slightly different requirements. To maximize reuse and minimize effort we are exploring the high-level interpretive approach using our Tcl binding to the toolkit. In addition, the APDE requires a graphical front-end. Tk provides a mechanism for graphics that is well integrated with Tcl. The remainder of the paper describes the various components of our requirements and our approach.

Tcl – Adding Flexibility to Applications

Developed at the University of California at Berkeley, Tcl (Tool Command Language) is an embeddable language library which can be linked to other applications. Unlike EXPRESS, Tcl is specifically designed for the direct construction and implementation of software. Tcl provides a fairly generic but reasonably high-level language.

The language is interpreted and resembles the UNIX shell in many ways. Elements are also derived from C and LISP. Despite its mixed heritage, much of the excess baggage from these other languages has been omitted leaving a modest but capable language. Tcl is described in detail by Ousterhout [16][17]. This section will only give a brief overview of the language and enough details to describe the sample scripts in the remainder of the paper.

The Tcl core consists of control flow statements such as `if`, `while`, and `case`. Tcl supports procedure definition, recursion, scoping, and other features typical of a high-level language. UNIX programs may be called and files manipulated. Expression evaluation is provided by a small set of primitives that manipulate strings. Conversion to and from other types is performed automatically.

The following Tcl fragment (from [16]) swaps the values of variables **a** and **b**, if **a** is less than **b**.

```
if {$a < $b} {
      set tmp $a
      set a $b
      set b $tmp
}
```

Here is a command to define a recursive factorial procedure:

```
proc fac x {
      if {$x == 1} {return 1}
      return [expr {$x * [fac [expr $x-1]]}]
```

```
}
```

The syntax and semantics are sufficiently close to C and the shell that the meaning of these examples should be intuitively obvious. For lack of space, we will not describe Tcl further. For that matter, it is not particularly germane to EXPRESS or to the EXPRESS Toolkit. Indeed, Ousterhout makes the point that the "*syntax of the Tcl language is unimportant: any programming language*" could provide similar features. The salient features of Tcl are that it is:

- programmable – Tcl applications are general-purpose and are not known in advance.
- efficiently interpreted – Tcl must be able to execute commands quickly enough that user interaction is not noticeably impeded.
- internally interfaceable to C – Tcl must allow one to bind existing C code to new Tcl commands that work synergistically with existing Tcl commands.

As the last bullet says, Tcl is designed to allow the addition of new commands. Our work adds several new commands to the Tcl language. The next section will describe these new commands.

Tcl Binding for the NIST EXPRESS Toolkit

This section describes a Tcl binding for the EXPRESS Toolkit. Note that:

- The binding is not meant to be complete. The binding currently exploits only a small fraction of the Toolkit's capability – primarily in the area of queries. It would be easy to extend the binding; however for now we have focused only on what we needed for the APDE.
- The binding is not meant to be definitive. Other bindings are possible for the same functions. We have not spent much time studying alternative bindings and do not claim that ours is better or worse than any others.

Schema – Loading new schemas

Using Tcl, we made new commands (bindings) for functions in the EXPRESS Toolkit. The primary command is "**`schema`**". The schema command loads EXPRESS files into the process so that they can be further manipulated.

The system is object-oriented in the sense that all of the objects in an EXPRESS schema become Tcl commands. Objects are named in a hierarchical way with "**`=`**" being used as a separator. For example, the schema **`s1`** is called "**`=s1`**" and the attribute **`a3`** in entity **`e7`** in schema **`1`** is named "**`=s1=e7=a3`**". This hierarchical notation is similar to the "**`.`**" separator used by other extensions of Tcl such as Tk widgets or the "/" separator used by the UNIX file system. The "**`=`**" has nothing to do with assignment or equality. It was chosen merely as a symbol that would not conflict with usages elsewhere in EXPRESS or in the Toolkit.

Basic Object Queries

All objects in a schema can be queried for the same information:

- Type
- File
- Line number

For example, given the entity **`$e`**, its line number can be retrieved with the following command:

```
$e -line
```

This could be used in a more complicated command such as the following:

```
puts "$e is defined on line [$e -line]"
```

Basic Scope Queries

Most EXPRESS objects have a scope which can contain other objects. For example, a schema can contain a number of entities and types. The objects immediately enclosed by another object may be listed with the **`-ls`** flag. This listing of objects may be further constrained with the additional flags **`-type`** or **`-glob`**.

The **`-type`** flag constrains the query so that only objects matching the given types are returned. For example, the following query returns all entities and types within the entity named by **`$e`**.

```
$e -ls -type "et"
```

The **`-glob`** flag is a simple constraint that does string matching based on the given glob-style pattern. For example, the following query returns all objects in **`$e`** which have names beginning with the letter "**`b`**".

```
$e -ls -glob "b*"
```

Object names returned by "**`-ls`**" are local names, without the **=** prefix. There is no need for the **=** prefix since that is precisely the name of the command used to make the query in the first place. However, queries can be made using the **=** prefixes just by using built-in Tcl commands. Because all objects have associated commands, Tcl's "**`info command`**" command may be used to return matching information. For example, the following command returns all of the Tcl objects whose name begins with the prefix **`=a`**:

```
info command =a*
```

More Queries

Complex objects have a nonobvious printable representation. For example, an entity may contain a list of subtypes, supertypes, types, other entities, etc.

This collection of information can be accessed piece by piece by using specific queries such as those shown elsewhere. Alternatively, objects can be queried for their whole printable representation. This is done using the **`-print`** flag.

```
$obj -print
```

The original printable representation is not explicitly saved. Rather, a printable representation is reconstructed from the internal representation.

Miscellaneous Commands

A number of miscellaneous commands exists. They are as follows:

Initialization

```
express_init
```

`express_init` initializes the system so that other EXPRESS Toolkit commands can be issued. Tcl, Tk, and other commands can be used before **`express_init`**. For example, the **`EXPRESS_PATH`** environment variable must be defined before the toolkit is initialized. This might be done as follows:

```
set env(EXPRESS_PATH) "public/APIB ~/myschemas"
express_init
```

Enabling Diagnostics

Certain toolkit operations cause processing that is not intuitive. For example, loading of a schema requires a complete parse and semantic analysis. It is possible to get some idea of what the toolkit is doing by having it print a brief description of each object and how it is being manipulated. This is done using the **`print_objects`** command.

The **`print_objects`** command is similar to the **`-P`** flag used by **`fedex`**. Objects are named by type. For example, the following commands cause the names of entities and types to be printed as they are processed:

```
print_objects e
print_objects t
```

Disabling Warnings

The toolkit generates warnings in certain instances. For example, certain EXPRESS constructions are not forbidden by the specification but are nonetheless unusual enough that they are likely wrong. These and others can generate warnings.

Warnings are disabled using the **`ignore_warning`** command. The command is called with the name of the specific warning as its argument. The following example disables the "downcast" warning:

```
ignore_warning downcast
```

Tk – Extensions to Perform X11 Graphics

Tk is an extension to Tcl. Written by John Ousterhout at Berkeley, Tk provides commands to manipulate X11 graphics. These commands are similar in style to those provided by the EXPRESS Toolkit binding.

We have incorporated the Tk extensions into our work so that we can create graphic displays with little effort and time.

As an example, the following script fragment creates a graphic image of a list of types in the current schema.

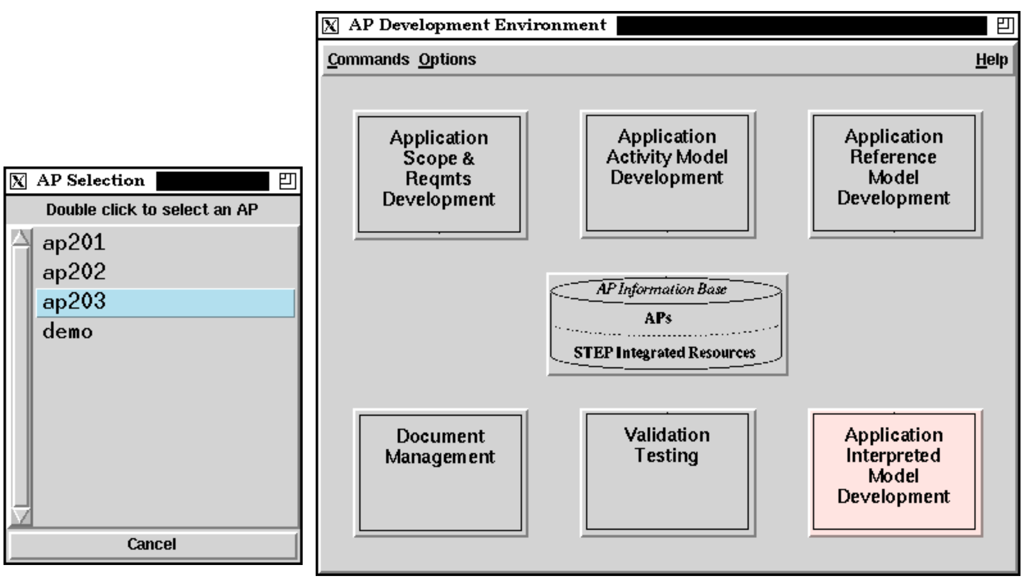
```
# create the listbox
listbox $listbox

# for each type, add it to the listbox
foreach type [$schema -ls -type "t"] {
      $listbox insert end $type
}
```

APDE – The NIST STEP AP Development Environment

The goal of the APDE is to provide an automated environment to facilitate the development of STEP Application Protocols and to improve their quality. An Application Protocol (AP) is the specification of product data structure in a particular application area. An AP is typically large (several hundred pages of documentation) and consists of various components including textual descriptions of data and data models. The APDE will provide a tightly integrated collection of NIST-developed and commercially-available software to allow AP developers to interactively create components of an AP and to store and retrieve APs, Integrated Resources, and other STEP-related documents and data models. The integrated tools will enable reuse of Integrated Resources and allow users to perform the various functions of AP development in a single, cohesive environment.

The following images are snapshots of different parts of the APDE.

The APDE opening screen shows a number of boxes that represent the primary activities in the AP development process. The menu bar has options or selections common to all activities. In the snapshot, the user has selected a particular AP (203) to work on and an activity (Application Interpreted Model Development) simply by clicking in the appropriate places.



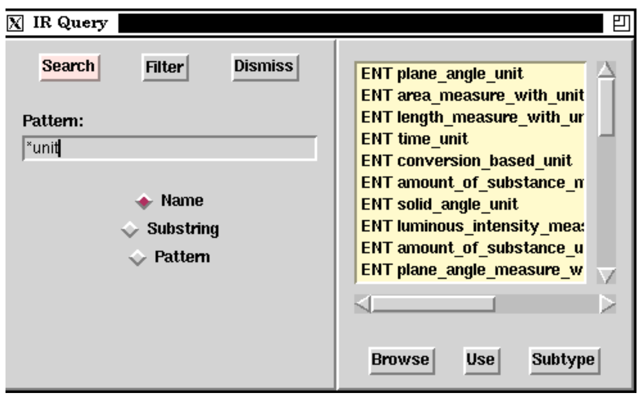
The **select** procedure creates and handles the AP selection browser. It is shown below. The select procedure does not do any queries against the toolkit. It merely gets the information from a prespecified list stored in the global variable **ap_list**. However, it suffices to show the basic techniques for building a small window of various components and doing simple interactions.

```
proc select {} {
      global ap_list ap_selection

      # Create a new window for the selection browser.
      toplevel .ap

      # Tell the window manager how to label it.
      wm geometry .ap +300+300
      wm title .ap "AP Selection"
      wm iconname .ap "AP Selection"

      # Tell the user how to use the browser.
      label .ap.text -text "Double click to select an AP"

      # Add a cancel button.
      button .ap.cancel -text Cancel -command {destroy .ap}

      # Create a place to display the list of APs
      # Pretty it up with a scrollbar and 3D effects.
      frame .ap.f -relief raised -bd 2
      listbox .ap.f.list -yscroll ".ap.f.scrollbar set" -font \
            10x20 -setgrid 1
      scrollbar .ap.f.scrollbar -command ".ap.f.list yview" \
            -relief raised

      # Add the actual AP names
      foreach ap $ap_list {
            .ap.f.list insert end $ap
      }

      # Make a double click select an AP and remove the browser.
      bind .ap.f.list <Double-Button-1> {
            set ap_selection [selection get]
            destroy .ap
      }

      # Place all the objects on the screen appropriately.
      pack .ap.text -side top
```

```
        pack .ap.f.scrollbar -side left -fill y
        pack .ap.f.list -side left -fill both -expand 1
        pack .ap.f -side top -fill both -expand 1
        pack .ap.cancel -side bottom -fill x -padx 2 -pady 2

        # Once everything is set up, wait for the user
        # to select an AP.
        tkwait variable ap_selection

        # Tell the APDE what AP the user has selected and return.
        config $ap_selection
    }
```

During the Application Integration Model Development activity, it is useful to be able to browse through EXPRESS objects in an Integrated Resource (IR). This is provided through an IR Query interface. The interface displays objects in the currently selected IRs which match according to various constraints such as types or patterns in the name.

The following snapshot shows the user querying for all the objects that end with the string “unit”.



The code that creates the IR Query interaction is too large to show in this paper, however a small excerpt will give the flavor of some actual queries against the EXPRESS Toolkit. Searching for a pattern will be described.

In order to search, the user must enter a pattern. The follow commands create the appropriate label, entry, and button:

```
    label .query.patt.label -text "Pattern: " -relief flat
    entry .query.patt.val -textvar itf(pattern) -relief sunken \
          -width 32
    button .query.search -text Search -command {
          ir_query "$itf(pattern)"
    }
```

Various elements of the user’s query are stored in the `itf` array. For instance, the user’s entry is stored in `itf(pattern)`. When the Search button is pressed, the pattern is passed to the procedure `ir_query` which does the actual searching.

```
proc ir_query {pattern} {
    global itf

    clear_result

    # figure out what kind(s) of symbols we're looking for
    set type ""
    if $itf(entity) {append type e}
    if $itf(type) {append type t}

    set found 0
    foreach schema [schema -ls] {
      # if schema is not currently selected, skip it
      if {!$itf(schema,$schema)} continue

      # add each matching object to the result
      foreach object [=$schema -ls -type $type -glob $pattern]
{
          add_result $object $schema
          set found 1
      }
    }

    if {!$found} {
      big_message "No matching definitions found"
    }
}
```

Unlike the **select** procedure shown earlier, the **ir_query** procedure avoids making direct Tk calls since the Tk objects are used by other procedures. Therefore, these uses are localized to procedures such as **add_result** (which adds a line to the display) and **big_message** (which pops up a window with an error in it).

The IR Query window offers a variety of other interactions. For instance, it is possible to see an object in its original context by clicking on its name and pressing the Browse button. This pops up an editor with the cursor on the specified object. This is initiated by the following button definition:

```
button .query.browse -text Browse -relief raised \
      -command browse_ir
```

The **browse_ir** procedure invokes **view_ir** on each selected object.

```
proc browse_ir {} {
    global short result apib sgml

    foreach sel [.query.list curselection] {
      set name [lindex [split [.query.list get $sel] " "] 1]
```

```
            view_ir $result($name) $name [
                string index [=$result($name)=$name -type] 0
            ]
        }
    }
```

The **view_ir** procedure invokes an IR browser. Our implementation of **view_ir** has changed frequently over time, first starting with a simple text editor and then trying a variety of SGML editors from different vendors. This is a good example of how Tcl provides ease in adapting the APDE.

The following definition is no longer used but shows a typically flexible approach. The procedure prefers to use an SGML editor but this is only possible if the IR can be found in our local SGML database which is not always the case. As a fallback, the user's text editor is invoked.

```
    proc view_ir {schema name} {
        global env

        # if there's an appropriate sgml file, view it
        if [file exists [set try [sgml_for $schema]]] {
          author_editor $try $name $class
          return
        } else {
          # fallback to a plain old text editor
          $env(EDITOR) [express_for $schema] [=$schema=$name -line]
        }
    }
```

Benefits of This Approach

We have constructed a binding so that the EXPRESS Toolkit can be controlled via the Tcl language. We have purposely designed the binding so that it is object-oriented and generally behaves consistently. Thus, there is very little to learn in order to use the binding.

Because Tcl is scripted, it is possible to create new Tcl code or modify old Tcl very rapidly. No time-consuming compile step is needed and the completed scripts are small. This is an extremely significant consideration. When using large libraries such as the Toolkit and the X Window system, even the smallest compiled programs are multiple megabytes in size and can take minutes to compile or just to relink. By comparison, our scripts are on the order of 1K to 2K and are not compiled or relinked.

Another benefit of the Tcl approach is reduced debugging effort. There is less code to look at so naturally, there is less to debug. More importantly, Tcl acts as a sort of firewall by separating the scripting from the C code. In the traditional approach, any piece of buggy C (or C++) code could corrupt any other piece of code, even a debugged piece. So a user application error could cause the toolkit internals themselves to misbehave. This makes for very difficult debugging. In comparison, the Tcl-controlled system does not permit user bugs to corrupt the Toolkit internals. Indeed, the user is prevented from corrupting any of the toolkits including Tcl or Tk.

Finally, all of the binding is high-level. The user does not have to worry about pointers. For example, objects are named mnemonically rather than with an opaque handle. With an opaque handle, the user could mistakenly pass an arbitrary pointer and the system would use it and fail. In contrast, if the Tcl user passes a meaningless name, it will be immediately rejected.

In general, all of the Tcl interfaces have similar benefits over their C counterparts. However, Tcl goes much further. Because of the design of Tcl, the user does not have to be concerned with memory allocation, hash tables, and many other low-level programming worries. Tk is particularly helpful in that it provides a convenient interface to the X Window System that is much simpler than direct calls into any of the existing widget libraries.

Summary and Conclusions

We have constructed a Tcl binding for the NIST EXPRESS Toolkit. This binding has enabled the rapid development of new tools including the APDE. The APDE environment was written very quickly, by gluing together a set of already existing tools with a small number of commands. In addition, we have taken advantage of other Tcl extensions such as Tk for controlling the X Window System.

We are happy with the end result. We can now write new applications and leverage existing applications much more quickly than before. The amount of time we spent writing the binding has easily been paid back in the time we have saved by working with it.

For More Information

Contact the Manufacturing Information Systems Division – National PDES Testbed (1-301-975-3386 or **`npt-info@cme.nist.gov`**) for more information about the software in general, or other NIST projects at the National PDES Testbed.

This software is a research prototype and is not presently packaged for distribution. When it becomes available, it will be obtainable through the automated source distribution server at the National PDES Testbed project. The server may be accessed via e-mail to **`nptserver@cme.nist.gov`**. If you are unfamiliar with the server, send the message "**`help`**" and you will receive an explanation of how to use it.

Acknowledgments

Design and construction of the software described in this paper is a part of the Application Protocol Development Environment (APDE) project which is funded jointly by the Computer-aided Acquisition and Logistic Support (CALS) program of the Office of the Secretary of Defense and by ARPA.

Thanks to Steve Ray and Josh Lubell for reviewing this paper.

The authors gratefully acknowledge Mary Mitchell, Lisa Phillips, Josh Lubell, Allison Barnard Feeney, and Jim Fowler for their contributions to the APDE.

Disclaimers

Trade names and company products are mentioned in the text in order to adequately specify experimental 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.

Both the application software and the server software are experimental. No claims are made for either. The software may change unexpectedly as we fix (or add) bugs. Esoteric behavior (such as disk full crises) will probably not ever be handled gracefully.

In no event will NIST be liable for damages, including any lost profits, lost monies, or other special, incidental or consequential damages arising out of the use or inability to use (including but not limited to loss of data or data being rendered inaccurate or losses sustained by third parties or a failure of the program to operate with programs not distributed by NIST) the programs, even if you have been advised of the possibility of such damages, or for any claim by any other party.

References

[1] Mason, H., ed., "Industrial Automation Systems – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles", Version 9, ISO TC184/SC4/WG PMAG Document N50, December 1991.

[2] Spiby, P., ed., "ISO 10303 Industrial Automation Systems – Product Data Representation and Exchange – Part 11: Description Methods: The EXPRESS Language Reference Manual", ISO DIS 10303-11:1992(E), July 15, 1992.

[3] The NIST STEP Part 21 Exchange File Toolkit: An Update, National Institute of Standards and Technology, Gaithersburg MD, NISTIR 5187, May 1993.

[4] Libes, Don, "The NIST EXPRESS Toolkit – Introduction and Overview", National Institute of Standards and Technology, Gaithersburg, MD, NISTIR 5242 (NTIS PB94-120664/AS), October 25, 1993.

[5] Libes, Don, and Fowler, Jim, "The NIST EXPRESS Toolkit – Requirements", NISTIR 5212, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993.

[6] Libes, Don, "The NIST EXPRESS Toolkit – Design and Implementation", *Proceedings of the Seventh Annual ASME Engineering Database Symposium*, San Diego, CA, August 9-11, 1993.

[7] Libes, Don, and Clark, Steve, "The NIST EXPRESS Toolkit – Lessons Learned", *Proceedings of the 1992 EXPRESS Users' Group (EUG '92) Conference*, Dallas, Texas, October 17-18, 1992.

[8] Libes, Don, "The NIST EXPRESS Toolkit – Obtaining and Installing", NISTIR 5204, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993.

[9] Libes, Don, "The NIST EXPRESS Toolkit – Using Applications", NISTIR 5206, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993.

[10] Libes, Don, “The NIST EXPRESS Toolkit – Programmer’s Reference”, National Institute of Standards and Technology, Gaithersburg, MD, to appear.

[11] Libes, Don, “The NIST EXPRESS Toolkit – Creating Applications”, National Institute of Standards and Technology, Gaithersburg, MD, to appear.

[12] Libes, Don, “The NIST EXPRESS Toolkit – Updating Existing Applications”, NISTIR 5205, National Institute of Standards and Technology, Gaithersburg, MD, June 9, 1993.

[13] Clark, S.N., “The NIST Working Form for STEP”, NISTIR 4351, National Institute of Standards and Technology, Gaithersburg, MD, November 1990

[14] Clark, S.N., “NIST STEP Working Form Programmer’s Reference”, NISTIR 4353, National Institute of Standards and Technology, Gaithersburg, MD, November, 1990.

[15] Clark, S.N., Feeney, Allison Barnard and Fowler, James, “Specifications for an Application Protocol Development Environment”, NISTIR 5248, National Institute of Standards and Technology, Gaithersburg, MD, August, 1993.

[16] Ousterhout, John, “*Tcl: An Embeddable Command Language*”, Proceedings of the Winter 1990 USENIX Conference, Washington, D.C., January 22-26, 1990.

[17] Osterhout, John, “*tcl(3) – Overview of tool command language facilities*”, unpublished manual page, University of California at Berkeley, January 1990.

Author Biographies

Don Libes is a computer scientist at the National Institute of Standards and Technology in Gaithersburg, Maryland, where he does research in manufacturing systems integration, interaction automation, and information dissemination and collaboration. Don has written over 75 computer science papers and articles including three books: *Life With UNIX*, *Obfuscated C and Other Mysteries*, and *Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs*. Don has received numerous awards including the International Communications Association Innovation Award and the Federal 100 for the development of Expect, a tool for automating and testing interactive applications.

Steve Clark is a computer scientist at Century Computing.