

Automating Interactive Applications in a Network Environment
by
Don Libes
National Institute of Standards and Technology
Gaithersburg, MD

ABSTRACT

Many programs demand to be run interactively. Word-processors are good examples, but many network applications (e.g., ftp, telnet) share the same fault. They cannot be run non-interactively.

Expect is a software tool designed to control interactive programs. Expect reads a script that resembles the dialogue itself but which may include multiple paths through it. Expect can run any program locally or remotely in order to automate a task.

Expect successfully deals with interactive programs and is particularly useful in a networked environment in which dissimilar machines must communicate. Expect solves the problem of "stick the password in the script" as well as several other long-standing problems with traditional work-arounds in these areas. Expect also provides the needed support for regression testing, network and computer load generation, and conformance testing in a networked environment.

In practice, Expect has entirely relieved numerous computer scientists and network managers of tasks that previously had to be performed by hand. The investment in writing Expect scripts is a minimal one-time cost. Expect is free and in the public domain. Expect is in use in hundreds of companies and universities in the U.S. and overseas.

Keywords: automation; communication; Expect; interaction; network;
Tcl; Tool Control Language

[End of abstract]

Footnote 1: Contribution of the U.S. Government. Not subject to copyright.

Footnote 2: 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.

INTRODUCTION

Many programs are designed strictly for interactive use. We call such programs "unautomateable". Editors are good examples, but many network applications (e.g., ftp, telnet) share the same characteristic.

Traditional scripting solutions, such as shells, command processors, and more specialized tools such as Kermit, are significantly limited in capability. For example, shells are restricted to very limited programming, such as straight-line algorithms because they cannot perform error checking from interactive programs. Kermit-like programs are designed specifically for communications on single-tasking systems and do not take advantage of local software or non-standard local communication interfaces.

In contrast, the research into Expect focused on the issues related to building a language for describing interaction. The result is a very general tool which surpasses older, traditional solutions.

Expect is a communication-independent system. It has a single generic interface for interacting with processes. Thus, Expect can make use of tools you have already. For example, if you have a security system or device driver or user interface, Expect will use it. Expect does not replace anything that you already have. It is simply a flexible and general, high-level control system.

Expect provides partial automation as well. It can automate part of an interaction and then let a human take over the interaction. For example, Expect can make all the network connections, and let the user type the password. Then Expect can take control again and finish the job. Control can be passed back and forth.

INNOVATION

Expect was the cross-product of the following three concepts:

- * a standard control language (Tcl) [2],
- * an appropriate set of concepts for interaction description and their implementation in Tcl,
- * the ability to manipulate other processes in a multitasking environment.

Unlike the traditional approaches, Expect stays away from building in specific I/O interfaces. Instead, Expect calls upon interactive and non-interactive processes that already exist. Expect has specific support for interaction and can thus drive programs in any way necessary.

This has several benefits:

- * Expect does not need to be configured for each system.
- * Expect can automatically use any I/O interface on a system.
- * Expect can control multiple processes and I/O simultaneously.

For example, Expect can create a connection to a remote system that uses an incompatible mail system, retrieve new mail back to the original system (or even a third system), and make it appear as if the two systems were compatible at a much higher level than they really are.

IMPLEMENTATION

This section briefly describes the implementation of Expect. A complete description is provided by [1].

Tcl - Tool Control Language

The Tcl core consists of control flow statements such as while, if/then, and others. Tcl supports procedure definition, recursion, scoping, and similar high-level functionality. Programs may be called and files manipulated. Expression evaluation is provided by primitives that manipulate the only primitive type - strings. (Conversion to and from other types is performed automatically, ala SNOBOL.)

The salient features of Tcl are that it is:

- * simple - It is expected that most Tcl programs will be short.
- * 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. (Tcl itself is written in C.)
- * internally interfaceable to other languages - Tcl must allow one to add new commands that work synergistically with existing Tcl commands.

As the last bullet says, Tcl is designed to allow the addition of new commands. Expect adds twenty-seven commands to the Tcl language. Tcl is defined by [2][3]. Expect is defined by [4]. Briefly, Expect provides:

- * send/expect sequences - "expect" patterns can include regular expressions.
- * standard, high-level language - Control flow (if/then/else, while, etc.) allows different actions on different inputs, along with procedure definition, built-in expression evaluation, and execution of arbitrary programs.
- * job control - Multiple programs can be controlled at the same time.
- * user interaction - Control can be passed from scripted to interactive mode and vice versa at any time. The user can also be treated as an

I/O source/sink.

Expect library

While Expect was being developed, Tcl provided a workbench with which to experiment. While Tcl suffices for most Expect use, it is possible to code directly in a compiled language. Expect comes with interfaces to C and C++. It is straightforward to interface to additional languages.

Multitasking, Pseudo-ttys

Expect requires a host system be multitasking. This is necessary in order for Expect to start other processes to work for it (much like a manager coordinates subordinates). Systems such as UNIX, VMS, and OS/2 are multitasking, while DOS is not.

Expect communicates with other processes using a technique commonly referred to as "pseudo-ttys" or "virtual terminals". A pseudo-tty is a logical abstraction of the hardware interface required to support interaction with a human, such as a keyboard and screen.

Pseudo-ttys allow programs to run unchanged even though they are not talking to physical hardware. Operations such as clearing the screen are not physically performed, but are nonetheless allowed and represented in the logical interface. Expect can thus detect when a program has requested or completed an I/O operation.

Combining these concepts provides the basis for Expect. A simple application is presented in figure 1. Under the direction of a script, Expect communicates with interactive programs as if it were a real person. Expect does this by internally spawning the process to be controlled. The I/O of the spawned process is then managed by Expect according to the script (figure 1a). Expect allows a person to take control from the script and return control at will (figure 1b).

PASTE FIGURE 1 IN HERE

Expect can also treat the real person like a process, thereby allowing the person to deal with a greatly simplified interaction. Any number of processes may be controlled. Figure 2 shows a typical instantiation with the person being treated as a process.

PASTE FIGURE 2 IN HERE

RESULTS

This section will briefly describe some performance aspects of Expect.

* Expect drives programs very quickly. In comparison to humans, response is instantaneous. Actual measurements of time and program size are described in [1] and [5].

* Expect never forgets a step. For instance, regression testing is very boring for humans. Once they have done a procedure before, they know that most of the time they're not going to see anything new. It's not surprising that they inadvertently skip tests. Expect's use in regression testing is described further in [6]

* Expect maintains security. Given a password, it will not record it anywhere. Thus, passwords cannot be exposed. Nor will a password be forgotten (unless it is so directed).

In practice, most Expect programs are relatively small. Many problems can be solved with only five to ten lines of Expect commands. Nonetheless, large programs are written, and the language scales well.

Learning how to use Expect is comparable to learning how to use any high-level language. It is possible to code applications in a matter of minutes. The most difficult part of any Expect task is getting a clear specification of another program's user interface. For example, while the Department of Defense File Transfer Protocol (FTP) is clearly specified, its corresponding user interface is quite vague, and implementations vary widely.

It is difficult to describe Expect without using superlatives. Indeed, the project has received hundreds of letters of thanks from national and international users. The number of notes received is remarkably large, particularly in view of the fact that users are under no obligation to write to the developer since Expect is free.

Expect can be applied to many problems [6][7], although this paper will only mention the narrow subset of interest to this audience. Here are some of the Expect applications and advantages that impact networking and communications. All of the applications described below have been implemented.

* Load Generation - Expect can be used to generate network or computer loads. This is useful in prototyping or pre-purchase network testing.

* Quality Assurance & Conformance Testing - Expect can run two programs simultaneously (or one against a standard) comparing outputs and timings for discrepancies. This is useful when verifying a new version of a program or interface functions the same as an old version. Regression and conformance testing are the core of much Expect use. See [6][8] for more detail.

* Faster, Easier Debugging - Expect can layer programmability onto a debugger, a network management tool, or any program. Most programs do not provide general program user interfaces. Expect provides a common programming interface to any extant tools.

* Faster Diagnostics - Expect scripts can be written to simulate everything that a user might do - for example, opening a connection to an out-bound modem, going through several switches, back through an in-bound modem, several logins and several applications. By testing these sequences frequently, we learn about and fix faults well before users have a chance to stumble onto them.

* Better Security - Expect scripts can automate everything except typing in passwords, allowing managers to know passwords but not worry about any of the technical responsibilities of them. Passwords

can be
batched, so that the passwords are entered at program initiation,
but
used much later. This works very well with all traditional
security
systems such as MIT's Kerberos.

* Password Maintenance - An Expect script can change passwords
for a
user who has accounts on multiple machines. The user types the
password once, while Expect does the rest. This reduces the
possibility of accounts ending up with differing passwords.

* Security Testing - Expect scripts can simulate humans to test
programs that are intentionally designed to be non-interactive.
Expect scripts can simulate hackers trying numerous passwords and
techniques to break into a system.

* Network and Host Integration - If Expect runs on one system, it
can
contact another system on which to work. For example, we have
Expect
controlling VMS, Cray, and Lisp Machines from a single UNIX
workstation. Remote systems do not have to run Expect.

* Presentation-level Conversions - Writing real network protocol
drivers for non-standard protocols is expensive. It is possible
to
write Expect translators in minutes, that work in real-time.

* Localize Remote Peripherals - Expect scripts can encode
commands to
connect to remote hosts and open remote devices, providing
subsequent
access as if they were local.

PLANNED ENHANCEMENTS

Porting

Currently, Expect runs on UNIX machines. It also runs on
machines
from small platforms (Intel 386) to supercomputers (Cray). We
know of
people running it on approximately 50 different vendors' brands
of
UNIX.

We anticipate ports to VMS, OS/2, and other multitasking
operating
systems in the near future. All of the environments already
provide
the base functionality that Expect depends upon.

Windows

Expect does not currently provide support for generalized bitmapped window systems. Several researchers are studying this area with experimental versions of Expect.

Availability

Since the design and implementation of Expect was paid for by the U.S. government, it is in the public domain. However, the author and NIST would like credit if this program, documentation or portions of them are used. Expect may be ftp'd as pub/expect/expect.shar.Z from ftp.cme.nist.gov. Expect will be mailed to you, if you send the mail message (no subject) send pub/expect/expect.shar.Z to library@cme.nist.gov.

REFERENCES

- [1] Libes, Don, "Expect: Curing Those Uncontrollable Fits of Interaction", paper and presentation, Proceedings of the Summer 1990 USENIX Conference, Anaheim, California, June 11-15, 1990.
- [2] Ousterhout, John, "Tcl: An Embeddable Command Language", Proceedings of the Winter 1990 USENIX Conference, Washington, D.C., January 22-26, 1990.
- [3] Ousterhout, John, "tcl(3) - overview of tool command language facilities", unpublished manual page, University of California at Berkeley, January 1990.
- [4] Libes, Don, "The Expect User Manual - programmatic dialogue with interactive programs", to appear as a NIST IR, National Institute of Standards and Technology, 1992.
- [5] Libes, Don, "Expect: Scripts for Controlling Interactive Processes", Computing Systems, Vol. 4, No. 2, University of California Press Journals, November 1991.
- [6] Libes, D. "Regression Testing and Conformance Testing Interactive

Programs", Proceedings of the Summer 1992 USENIX Conference, San Antonio, Texas, June 12-15, 1992.

[7] Libes, Don, "Using Expect to Automate System Administration Tasks", paper and presentation, Proceedings of the Fourth USENIX Large Installation Systems Administration Conference, Colorado Springs, Colorado, October 17-19, 1990.

[8] Woodson, Brian, "Regression Testing Using Expect", Quality In Software Conference, Santa Clara Valley Software Quality Association, June 29, 1991.