Proceedings of FREENIX Track:
2000 USENIX Annual Technical Conference
San Diego, California, USA, June 18–23, 2000

THE AT&T AST OPENSOURCE SOFTWARE COLLECTION

Glenn S. Fowler, David G. Korn, Stephen S. North, and Kiem-Phong Vo

The AT&T AST OpenSource Software Collection

Glenn S. Fowler, David G. Korn, Stephen S. North and Kiem-Phong Vo
AT&T Laboratories – Research
180 Park Avenue, Florham Park, NJ 07932, U.S.A.
gsf,dgk,north,kpv@research.att.com

Abstract

This paper introduces a large collection of reusable software components that AT&T is making available in an OpenSource form. This software has been widely used around the world and includes well-known components such as KornShell, Nmake, Graphviz, Sfio, Vmalloc and Cdt.

1 Introduction

AT&T is not a newcomer to the UNIX market. In fact, it is where UNIX was born. However, AT&T is a newcomer to the world of OpenSource. This paper highlights our entrance into that domain.

UNIX was first invented when AT&T comprised the entire Bell Telephone System including Bell Telephone Laboratories, Western Electric and 23 Regional Operating Companies. This early AT&T company was a regulated monopoly restricted to the telecommunications business and prohibited from entering other businesses. As such, all inventions from Bell Telephone Laboratories unrelated to the telecommunications business could only be licensed externally. The UNIX system fell into this category and was licensed early on for academic use.

In the late '70s and early '80s, AT&T wanted to expand its business into new markets including the computer market. In exchange for this right, AT&T agreed to the now famous consent decree that split off the various Regional Bell Operating Companies or RBOCs. Bell Telephone Laboratories was divided into two parts, AT&T Bell Laboratories and Bellcore (subsequently renamed Telcordia) which performed research and development for the RBOCs. The UNIX system remained with AT&T Bell Laboratories and was commercialized for the first time. With the entry into the computer market, AT&T also became more restrictive in the release of potentially commercializable software.

In the early '90s, AT&T realized that it could not compete effectively in the computer business and began to refocus exclusively into its core strengths, the communications business. As a result, the UNIX division was sold to Novell. In the subsequent "trivestiture," the company split off the computer division, NCR, and further separated the communication business two independent companies, Lucent Technologies for equipment and the current AT&T for services. AT&T Bell Laboratories was divided into two parts, Lucent Bell Laboratories and AT&T Laboratories. All copyrighted software at trivestiture time remained available to both Lucent Bell Laboratories and AT&T Laboratories after the split.

Our software research department, the Advanced Software Technologies department (AST), was formed in the late '70s in the original Bell Telephone Laboratories. It was separate from both UNIX research and development organizations. Our charter was to improve the productivity of AT&T software development. We achieved this by inventing new algorithms and techniques for existing APIs, creating new powerful APIs in the forms of libraries, languages and tools, and developing techniques for accurate build and configuration of software. The audience for our software was the myriad development organizations in AT&T. Since these organizations used diverse equipments and operating systems, a key part of our work was to develop techniques to enhance portability.

Our tools and libraries spread quickly among the AT&T development organizations. To help with internal technology transfer, a separate organization from research was set up to handle support and distribution. Unfortunately, external release of our software was not as smooth. The UNIX development group was responsible for outside release and they were often reluctant to adopt software that they

did not develop. There were, however, notable exceptions such as KornShell, Curses, and Malloc. The latter two libraries eventually became standard parts of Solaris, Irix and Unixware, UNIX variants derived from the System V Release 4.0 UNIX version.

Events turned worse for internal technology transfer after the sale of UNIX to Novell. To reduce the cost for software support and to ensure compliance with development methodologies such as ISO 9000, AT&T development organizations were pushed to buy vendor supported software. In this environment, the most effective avenue to transfer technology from research to development was to license the software to external vendors who, in turn, resold to AT&T.

The recent advance of the OpenSource movement, with the inception of the Open Source Initiative (OSI), opened a new venue for making research software widely accessible. From our point of view, the primary advantages to OpenSource are:

- Increased influence on national and international standardization efforts,
- Creating and supporting alternatives to closed systems,
- The ability to attract vendors to distribute and support the software,
- Improved software quality due to widespread use, and
- Increased visibility of AT&T Laboratories in the research community.

These benefits fit well with the current business directions of AT&T which emphasize on building the communication services business, not selling software. Thus, AT&T intellectual property managers became more open to arguments in favor of an OpenSource software release based on the set of principles called the Open Source Definition as specified by the OSI on the website:

`http://www.opensource.org/osd.html`

In early 1999, we started an effort to release much of the software developed in research under an OpenSource license. After several months of discussions and many rounds of negotiation, AT&T has successfully produced a license that we believe meets the conditions for OpenSource certification and have submitted the license to OSI for official certification.

The AT&T AST OpenSource software represents many years of effort and covers a broad set of libraries and applications. Much of the early work was described in the book Practical Reusable UNIX Software[2]. The software includes a large subset of the POSIX utilities including the latest version of KornShell. In addition, there are many libraries and utilities not available elsewhere. The software is portable across virtually all UNIX environments, including OpenEdition on MVS (which uses EBCDIC) and Windows systems given a suitable UNIX layer such as UWIN[13].

It is neither possible nor appropriate to describe all the software components here, so the remainder of the paper will focus on our approach to building and packaging software and the terms of our OpenSource license. Interested readers can check the given references for details on particular software components.

2 Libraries

The AT&T AST OpenSource software collection consists of is upward of 3/4 million LOCs. Despite this large size, most of the software was created by a small group of researchers, about 6 at its peak. The software embodies some of the most powerful algorithms and data structures known. For example, the Graphviz package implements our patented graph drawing algorithms that automatically generate pictures of directed and undirected graphs with thousands of nodes and edges in seconds. A focus in our work is to make such algorithms and data structures widely reusable, not just in the tools that we create but also in applications that others would write. Another focus for our software is portability. We wish to make it easy to build applications that transparently compile and execute on at least all different UNIX platforms and certain selected others. The limited human resource and the desire for wide software reuse led us to conclude that:

> A major part of our software development effort should be directed toward the creation of powerful reusable libraries that would enable other tools and applications to be built by simply assembling such libraries.

Thus, we embarked on a program to write software libraries that encompass core computing functions such as I/O and memory allocation and other new algorithms and data structures such as data compression and differencing and graph drawing. Below is a partial list of libraries available in the collection:

- *Libast:* This is the porting base library[8] for our software tools. It includes a common header

that provides common data types such as `size_t` and others that may be missing on a particular platform. Similarly, functions are provided to fill in missing ones (e.g., `bcopy()` on non-BSD systems) and to replace existing ones that are inefficient. Libast also provides new convenient functions such as `strperm()` to convert a **chmod** file mode expression into a `mode_t` value.

- *Sfio:* This I/O library[9] provides a robust interface and implements new buffering and data formatting algorithms that are more efficient than those in the standard I/O library, Stdio. For backward compatibility, Sfio also provides emulation code for Stdio that is suitable for both recompiling and relinking Stdio applications.

- *Vmalloc:* This memory allocation library[22] allows creation of different memory regions based on application-defined memory types (heap, shared, memory mapped, etc.) and some library-provided memory management strategies. A backward-compatible Malloc interface is provided that additionally allows an application to selectively perform memory debugging or profiling by setting environment variables.

- *Cdt:* This container data type library[23] provides under a unified interface a comprehensive set of containers: ordered/unordered sets/multisets, lists, stacks and queues. These container data types are based on efficient data structures such as hash tables and splay trees.

- *Libexpr:* This library provides run-time evaluation for simple C-styled expressions. It forms the basis for commands such as **tw**[7], a file tree walker and **cql**[4], a flat file database language.

- *Libgraph:* This graph library[18, 11] supports attributed graphs, generalized nested subgraphs, and stream file I/O in a flexible graph data language. It is built on top of the Cdt library and employs disciplines for I/O, memory management, graph object namespace management and object update callbacks. This library is the base of the Graphviz package to be discussed later.

2.1 Design considerations

Desirable qualities of a reusable library include applicability, efficiency, ease of use and ease of maintenance. However, there is no simple set of rules that can guarantee the simultaneous achievement of these qualities. The design considerations below serve as guiding principles in our work:

- *Necessity:* This is at the heart of the idea of focusing on writing libraries before applications. In developing tools and applications, we first look into how they can be structured as one or more library functions. In this way, any resulting new APIs would be guaranteed to fill some specific needs and not just arise out of some academic exercise. For example, our versions of the POSIX commands were written first as library functions with the final commands being just simple drivers to parse command line options and make the appropriate function calls. Many of these functions are then reusable as built-ins in other applications such as the shell.

- *Generality:* Each library attempts to encompass as much functionality as possible without sacrificing efficiency. This sometimes means unifying separate but related concepts under a single uniform interface. A good example of this is the memory allocation library, Vmalloc, that unifies under a single interface various memory allocation policies including methods to debug memory errors and to profile memory usage.

- *Consistency:* We make sure that our libraries and functions follow the same interface structures. For example, many libraries provide functions to create structures to store states and return handles referring to such structures. In turn, these handles are used in future function calls to access stored resources. For consistency, we insist that the handle is always the first argument in such calls. This should be contrasted, for example, with Stdio where a `FILE*` handle can be either the first or the last argument.

- *Efficiency:* A reusable library is intended to be used in many applications so performance is a key to its success. This means that its implementation should use the best available algorithms and data structures. However, this is not enough since different applications may have different needs that require specific adaptations. We developed a discipline and method library architecture[24] to unify different algorithms and techniques under a single interface while still allowing applications to tune for efficiency.

- *Modularity:* Modularity is the key in easing maintenance effort as it reduces the interdependency among different libraries as well as among

functions within the same library. To the extent possible, different libraries are kept independent from one another. However, there are exceptions such as memory allocation or I/O that must be negotiated by Vmalloc or Sfio.

- *Irredundancy:* Along the same line of easing maintenance effort, most algorithms and techniques are implemented once in one component, and all other components refer to this one implementation. In this way, we never need to make changes in multiple places for a single algorithmic modification.

- *Extensibility:* Extensibility means the ability to add or change features without breaking existing application code. Most of our libraries are based on the discipline and method architecture[24]. Where applicable, a user-defined discipline structure provides a version field that must be initialized by its caller. Thus, a new library implementation can then use the version field to detect the caller's vintage and act appropriately. The Sfio extended print discipline[10] uses this technique so that a new version can support an obsolete feature for at least one generation before discarding it. In turn, this gives application developers time to adapt their code.

- *Robustness:* Robustness means (1) thoroughly testing functionality, (2) keeping the code free from artificial constraints such as fixed size arrays or integer sizes, and (3) avoiding unsafe interfaces. Dynamic memory allocation is judiciously used to construct any required data structures. An example of the last point is our Sfio function `sfgetr()` that replaces the infamous Stdio `gets()` function and removes any concern about buffer boundary violation. Each of our libraries comes with a comprehensive regression test suite that was built over time based partially on bug reports.

- *Portability:* A basic goal for us is to be able to run our code on all platforms that we have access to. This includes all UNIX/Linux platforms and others such as Windows and MVS. However, portability means more than just that. Our software is configured using the **iffe**[6] approach to target local platform features and maximize performance. For example, the Sfio library selects between memory mapping and other I/O system calls by running a performance test at build time.

3 Tools

Aiming at an efficient, easy to use, and portable computing platform, we have reimplemented nearly all POSIX command tools. We also invented a number of new tools some of which exerted influence on the standards. For example, our KornShell language helped to define the POSIX 1003.2 specification for shell language and our Pax tool for file packaging has been included in POSIX 1003.2. Our tools provide a wide range of functionality. Below are a few examples:

- **nmake**[3]: A far more powerful *make* language that supports dynamic dependency generation and a higher level specification language.

- **iffe**[6]: We use **iffe** to handle architecture-specific features so that such information can be specified in the library source code instead of the makefiles. **iffe** detects architecture-specific features similar **autoconfig**, but at a much more localized level. For example, most GNU packages have a single `config.h`, whereas the Libast library has 25 **iffe** files. By localizing the configuration tests we can limit the amount of code that must be recompiled when individual **iffe** configuration scripts change. Unlike **autoconfig** and **old-make**, **nmake** handles all **iffe** file generation and dependencies automatically as part of the build process.

- **tw**[7]: This command replaces **find** and **xargs** and provides more general searching.

- **cql**[4]: This command provides a C database query language that works on both flat file and binary databases. It performs better than **awk** or **perl**.

- **3d**[5]: This command is a combination of a shell script and a shared library. It modifies the file system semantics to enable viewpathing. That is, different file trees can be virtually overlaid on top of one another with a copy-on-write semantic to the higher layer.

- **warp**: This command is implemented similarly to **3d**. It can be used to run a process as if the time were set to some specified time and the clock speed was set to some specific rate. It was useful for Y2K testing.

The tool development follows a few principles that help to keep the tools uniform, portable, and robust. We discuss these principles next.

- *Conformance to standards:* Our tools conform to the 1992 IEEE POSIX 1003.2 standard[20] and provide additional extensions where appropriate. Thus, at the least, strictly conforming POSIX applications can use our code without change. We have rewritten most of the tools in the base standard and User Portability sections except for a few such as **awk**, **diff**, **mailx**. We imported the GNU version of **diff** and the BSD version of **mailx** and enhanced the latter to add features such as MIME enclosures and IMAP.

- *Conformance to common conventions:* On various platforms, many common utilities provide extensions that are beyond the standard but well liked by users. Whenever possible, our versions of the utilities provide the same extensions. For example, most of the utilities accept the long name options that GNU supports. To handle conflicts between extensions done in different universes such as BSD or System V, we added a configuration option **UNIVERSE** to control the behavior of such utilities. In addition, the **getconf** utility is extended so that users can set configuration parameters such as **UNIVERSE**.

- *Avoiding absolute pathnames:* One of the banes of binary distributions is embedded path names. For example, the **file** command may consult **/etc/magic** for file descriptions. Its not appropriate for the our **file** command to look in the same place. Instead of adding a new environment variable for each candidate fixed path, we use the `$PATH` environment variable, which must be set properly to use any package, to locate command related files. The convention is: the files for command *foo* are found by doing a `$PATH` search for the `../lib/`*foo* directory. A binary directory tree can be installed anywhere without recompilation; the only requirement is that the binary `bin` directory is added to `$PATH`.

- *Avoiding size restrictions:* Our tools are freed artificial constraints such as fixed size arrays or small file sizes. For example, even though the POSIX standard only requires handling of text files with line lengths 2K or less, our tools do not have such line length limits. Instead, they use the record-reading function `sfgetr()` in the Sfio library which allocates dynamic memory as necessary to buffer lines with arbitrary lengths. Via the use of Sfio, our tools also transparently work on systems that support files with offsets larger than 32 bits.

- *Combining related utilities into one:* Whenever multiple related utilities can be combined into one, we do so. For example, our **pax** command combines the functionality of **cpio** and **tar**. In addition, it supports multiple formats such as ANSI and EBCDIC standard labelled tape, VMS backup and Microsoft cabinet files and also compression methods such as compress, gzip, and our own Vdelta[12] method for compression and differencing. Other notable examples include the unification of **grep**, **fgrep**, and **egrep** in a single command, the support of MIME Base64 and Binhex encodings in **uuencode** and **uudecode**, and the support of MD5 hashing in **cksum**.

- *Self-documenting tools:* Our tools use a common library function `optget()` to parse command line arguments. We extended this option parser so that it can generate the man page in one or more formats. For example, `pax --man`, `pax --nroff`, and `pax --html` would generate the documentation on the screen, in Troff, or in HTML format.

- *Building and using powerful reusable libraries:* It is worth emphasizing again here that our main focus is on building powerful reusable libraries that can be easily assembled into commands. This helps to maximize the reusability of the code. For example, even the KornShell itself was implemented as a library. In turn, this enabled the creation of **tksh**[14], a combination of shell and the Tk graphics library[19].

4 Graph visualization

A large part of our work is in software reegineering and data visualization. To help with this effort, we developed Graphviz[11], a collection of portable tools for rendering and interacting with abstract graph (network) drawings.

The main Graphviz layout programs, **dot** and **neato**, read text specifications of the nodes and edges of a graph, and emit drawings in a graphics language such as Postscript, pic, GIF, Metapost, VRML or PNG. We have combined these rendering engines with the graphics editor **lefty**[15] to build an interactive diagram manipulator **dotty**[16]. To provide support for a more popular scripting environment, Graphviz can be compiled as a Tcl/Tk extension.

Graphviz was also made into a web server by adding GIF and PNG drivers and a wrapper script

to run **dot** or **neato** as a remote cgi-bin service. This service is now employed within AT&T and by outside projects (for example, mozilla.org) and represents a simple experiment in how programs providing lightweight services can be reused more easily as web services than as packages that must be manually downloaded, installed and kept up-to-date. Also, for more sophisticated user interface customization we created Grappa, a Java graph library that communicates with **dot** as a server in the same way as **lefty/dotty**. Grappa can run standalone or as an applet.

Recent work on Graphviz addressed dynamic layout, where diagrams are maintained on-line with stable incremental updates. This work required devising event-based APIs and modifying the front ends to handle layout event streams instead of batch layout. The OpenSource code release includes a component addressing the Microsoft Windows platform. Our goal here was to create a fully OLE-aware network diagram editor. The editor is embeddable, may contain foreign objects as content and can be controlled via C++ or scripting languages such as Visual Basic. To achieve this, we factored the diagram editor into a generic OLE client-server, Montage[25], that provides UI management, persistence for non-hierarchical collections of objects, and domain-specific components to interpret user interface events or manage layouts. This facet of the Graphviz project diverged a great deal from our UNIX roots. However, Montage is a valuable contribution to the OpenSource community. To our knowledge it is the only fully OLE-aware software component available in an OpenSource form, and perhaps the only one outside of Microsoft, Visio, and Borland/Inprise. Moreover, the container library is a separate, clean design well suited to reuse, not laden with application-specific semantics.

5 Packaging

We employ a packging process designed to ease the reproduction of our environment on multiple platforms. This process provides a mechanism for bundling a set of source components, transporting them to another platform, making binaries from source using the native compilation system, and bundling binaries for use on other equivalent platforms.

5.1 Packages

The smallest distribution unit is a *package*, i.e., a collection of source or binary *components*. A *source component* is a group of source files controlled by a *makefile*; the makefile and all related source reside in a directory named by the component. *Making* a component generates binaries from the source and makes them available for use. A *binary component* contains all of the generated binaries corresponding to a source component. Most components generate commands or libraries and interface files, but some may provide only documentation or data.

A component may depend on other components. These dependencies define a *make order*. That is, a component may require that other components be made before it is made. Packages might also have dependencies. For example, the Graphviz package requires the Libast package.

The traditional method to handle component make order to hard-wire the order in package makefiles or build scripts, with the restriction that each package reside in its own directory hierarchy. This method defeats any attempts at sharing code between packages. For example, the GNU *fileutils*, *findutils*, and *textutils* packages each have a **lib** source directory. Out of a combined 152 files, 53 are unique to one package, 30 are shared between two, and 13 are shared between all three. This is an unacceptable situation and inevitably leads to duplication and splintering – maintenance nightmares for a small group like ours.

nmake solves the component make order problem. Given a collection of component makefiles, **nmake** constructs a component dependency graph, and makes the components in order, using a separate **nmake** invocation for each component. Independent components are detected to facilitate concurrent makes. It is important to note that when components are added to a package hierarchy the new components are automatically detected and made in the proper order. This means that, barring generated binary name clashes, packages and components can be freely and safely combined.

5.2 Versioning

As mentioned, there are two types of packages: source and binary. These come in two flavors: *base* and *delta*. A *base* package contains a complete copy of all the package components. A *delta* package is similar to a *patch*: it contains only the differences from its base package. Unlike a patch, delta differences are maintained at a byte level instead of a text line-by-line level. This allows binary deltas as well as source deltas. Delta packages also contain *delete* information, so that files may be deleted from a component as it ages. Packages are simply compressed

archive files maintained by our **pax** command which computes deltas using our `vdelta` algorithm. Support files are simply added to the archive of component source as the packages are generated.

Delta packages form the basis of a simple but complete version management strategy. With a little discipline we are able to record and document software updates and bug fixes. Each component has a **RELEASE**, **CHANGES**, or **CHANGELOG** file that contains a dated comment line for each notable change, in reverse chronological order (newest entries at the top.) After component source is modified and tested (including **RELEASE** file edits), a delta source package is made to record all of the changes. Delta source packages are quite efficient in space and typically take up less than 1 percent of the corresponding base source packages.

Each package is stamped with its creation date and delta count (starting at 1). As a package archive is written, its stamp and the stamps of packages it depends on are recorded as files in the archive. When unpacking a package a simple sort on the stamp files tells if the proper dependent packages are present. Since backwards compatibility is guaranteed, the stamp checking rule is simple: any package stamp equal to or newer than the requested stamp is acceptable.

5.3 Source vs. Binary

The package layout maintains source files (readonly) in a separate directory tree from binary files (generated). This feature allows many binary packages to be made from a single source copy, handy when the package directory tree is cross-mounted on hosts with different architectures. Files in the binary directory tree take precedence over files in the source tree. So local modifications can be simply done by copying source files to the binary tree and modifying them there. In this way, the original and modified files can be compared, and changes made for one architecture won't interfere with the others. A source delta (patch) is simply made by recording the differences between source files in the binary and source trees.

5.4 The package Command

The **package** command is the interface for all package management. This command is part of the **INIT** package that all other packages require. The **INIT** package must first be downloaded into an empty directory tree and installed (`gunzip <` `INIT-`*yyyy-mm-dd* `| tar xvf -`). Then other packages can then be downloaded, made, and/or installed by the **package** command.

Each package has a description file, *package*`.pkg` (an **nmake** makefile), that lists its components and package dependencies. Any component described by an **nmake** makefile can be part of a package; no other files or auxiliary package information is required. The package is used for the following operations:

- `write [ base|delta ] [ binary|source ]` *package*: This creates an archive for *package*, including version stamp and binary checksum support files.
- `read [` *file*`.tgz|`*file.nnn* `]`: This reads the package base archive *file*`.tgz` or package delta archive *file.nnn*.
- `make [` *package* `]`: This makes and installs the binaries for *package*, or all packages if *package* is omitted.
- `verify [` *package* `]`: This verifies the installed binaries for *package*, or all packages if *package* is omitted, against the package checksum files.
- `test [` *package* `]`: This runs the regression tests for *package*, or all packages if *package* is omitted.
- `use [` *uid*`|`*package* `]`: This runs an interactive shell with environment initialized for using *package* or the package installed by the user *uid*. An unfortunate side effect of using shared libraries (DLLs) is that some systems require specific (and different for every system) environment variable settings to properly locate the DLLs at runtime.

6 License terms

The AT&T Source Code agreement was written both to satisfy the Open Source Definition, and to protect AT&T's intellectual property and other rights. Before creating a new license, we carefully reviewed the main licenses already in use, particularly, GPL, LGPL, QPL, Apple Public Source License, and the IBM open source license. These were not satisfactory to AT&T as, for example, they do not adequately cover patent rights. Often such holes are just as detrimental to licensees as they are to the licensor.

The AT&T Source Code Agreement (ASCA 1.2D), listed in the Appendix, gives licensees the right to:

- Read, study, display, compile, and execute binaries made from the source code.
- Use AT&T patents in the original code to execute original or modified software.
- Redistribute the original source package in any format, as long the contents are preserved exactly.
- Distribute patches that include the copyrighted source code.
- Distribute binaries made from original or modified source code.

From a licensee's standpoint, the main conditions are:

- Distribution can only be made to those that agree to the license terms,
- If modifications to the source are made public, then AT&T has the right to include these changes in its package, and
- AT&T controls the contents of the official source distribution.

In many other ways the license terms are liberal toward commercial OpenSource licensees. The ASCA allows licensees to charge for redistribution. It also grants the right to use AT&T patents involved in the code, even when modified.

Some of the concerns brought up in the OpenSource review include those below. We felt that these license terms were reasonable, and left them in.

- A restriction against framing the AT&T website, perhaps to suggest a relationship with AT&T.
- Your rights under this agreement can be terminated automatically if we receive a non-frivolous claim of a patent infringement by a third party related to the source code on our website. In the event of an infringement claim, you would have to replace any infringing portion of the source code with non-infringing code or license the patent from the third party. For this reason, the agreement requires you to periodically check the AT&T website for such infringement notices.
- A "no strict construction" clause. This is an agreement that the license should not somehow be construed literally against what was really intended, to the detriment of either party.

Some very significant improvements were made as part of the OpenSource negotiation, for example, plugging holes in the license concerning the equivalent of "fair use" (since the ASCA relies on granting specific rights, instead of transferring ownership of a copy of a copyrighted work) and avoiding cumbersome restrictions on charging for redistributed copies, shrink-wrap licensing and repackaging of the source, among others. The AT&T OpenSource license agreement can also be viewed at:

`http://www.research.att.com/sw/license/ast-open.html`

7 Conclusion

This paper introduced the AT&T AST OpenSource software collection. This software collection is nearly twenty years in the making and includes tools such as the KornShell language, the Nmake system, the Graphviz package for graph drawing, and core computing and algorithm libraries such as Sfio, Cdt and Vmalloc. Many of the components were previously available for non-commercial use from the website:

`http://www.research.att.com/sw/tools/`

In just the past year and a half, more than 40,000 copies have been downloaded. Many large and mission-critical projects both within AT&T as well as around the world are dependent on these components. A frequently asked question from external users is "How do we license the software for production use?" This OpenSource release provides an answer.

Acknowledgments

The authors thank Ben Lee and Tom Restaino for legal work, and Dave Belanger, Jeff George and David Nagel for backing the Open Source release.

References

[1] ANSI. *American National Standard for Information Systems – Programming Language – C.* American National Standards Institute, 1990.

[2] Edited by B. Krisnamurthy. *Practical Reusable Unix Software.* John Wiley & Sons, Inc., 1995.

[3] Glenn S. Fowler. The Fourth Generation Make. In *Proc. of the USENIX 1985 Summer Conference*, pages 159–174, 1985.

[4] Glenn S. Fowler. cql – A Flat File Database Query Language. In *Proc. of the USENIX Winter 1994 Conference*, pages 11–21, January 1994.

[5] Glenn S. Fowler, David G. Korn, and Herman C. Rao. n-DFS: The Multiple Dimensional File System. *Trends in Software: Configuration Management*, 2, 1994.

[6] Glenn S. Fowler, David G. Korn, J.J. Snyder, and Kiem-Phong Vo. Feature-Based Portability. In *Proc. of the Usenix VHLL Conference*, pages 197–207. USENIX, 1994.

[7] Glenn S. Fowler, David G. Korn, and Kiem-Phong Vo. An Efficient File Hierarchy Walker. In *Proc. of the Summer '89 Usenix Conference*, pages 173–188. USENIX, 1989.

[8] Glenn S. Fowler, David G. Korn, and Kiem-Phong Vo. Principles for Writing Reusable Library. In *Proc. of the ACM SIGSOFT Symposium on Software Reusability*, pages 150–160. ACM Press, 1995.

[9] Glenn S. Fowler, David G. Korn, and Kiem-Phong Vo. Sfio: A Buffered I/O Library. *Software—Practice and Experience*, Accepted for publication, 1999.

[10] Glenn S. Fowler, David G. Korn, and Kiem-Phong Vo. Extended Data Formatting Using Sfio. In *Proc. of the 2000 Usenix Conference*, USENIX, 2000.

[11] E. R. Gansner, E. Koutsofios, S. C. North, and Kiem-Phong Vo. Graph Visualization in Software Analysis. In *Proc. of the Symp. on Assessment of Quality Software Development Tools*, pages 226–237, 1992.

[12] James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. An Empirical Study of Delta Algorithms. In *IEEE Software Configuration and Maintenance Workshop*, 1996.

[13] David G. Korn. Porting UNIX to Windows NT. In *Proc. of the 1997 Usenix Conference*. USENIX, 1997.

[14] Jeffrey L. Korn. Tksh: A Tcl library for KornShell. In *Proc. of the USENIX Tcl/Tk Workshop*, pages 149–159, Monterey, CA, July 1996.

[15] Eleftherios Koutsofios and David Dobkin. Lefty: A two-view editor for technical pictures. In *Proc. of Graphics Interface '91*, pages 68–76, 1991.

[16] Eleftherios Koutsofios and Stephen C. North. Applications of Graph Visualization. In *Proc. of Graphics Interface 1994 Conference*, pages 235–245, Banff, Canada, May 1994.

[17] David R. Musser and Atul Saini. *STL Tutorial and Reference Guide*. Addison-Wesley, 1995.

[18] Stephen C. North and Kiem-Phong Vo. Dictionary and Graph Libraries. In *Proc. of the Winter '93 Usenix Conference*, pages 1–11. USENIX, 1993.

[19] John K. Ousterhout. *Tcl and the Tk Toolkit*. Reading, MA, 1994.

[20] POSIX. *IEEE Standard 1003.2–1992, ISO/IEC 9945-2:1992, POSIX – Part 2: Shell and Utilities*. Institute of Electrical and Electronics Engineers, 1993.

[21] Robert Sedgewick. *Algorithms, 2nd Edition*. Addison-Wesley, 1988.

[22] Kiem-Phong Vo. Vmalloc: A General and Efficient Memory Allocator. *Software—Practice and Experience*, 26:1–18, 1996.

[23] Kiem-Phong Vo. Cdt: A Container Data Type Library. *Software—Practice and Experience*, 27:1177–1197, 1997.

[24] Kiem-Phong Vo. An Architecture for Reusable Libraries. In *Proc. of the 5th International Conference on Software Reuse*. IEEE, 1998.

[25] Gordon Woodhull and Stephen C. North. Montage – an ActiveX Container for Dynamic Interfaces. In *Proc. of the 2nd USENIX Windows NT Symposium*, pages 109–116, Seattle, CA, August 1998.

AT&T AST OpenSource SOURCE CODE AGREEMENT

Version 1.2D

PLEASE READ THIS AGREEMENT CAREFULLY. By accessing and using the **Source Code**, you accept this Agreement in its entirety and agree to only use the **Source Code** in accordance with the following terms and conditions. If you do not wish to be bound by these terms and conditions, do not access or use the **Source Code**.

1. **YOUR REPRESENTATIONS**

    1. You represent and warrant that:

        a. If you are an entity, or an individual other than the person accepting this Agreement, the person accepting this Agreement on your behalf is your legally authorized representative, duly authorized to accept agreements of this type on your behalf and obligate you to comply with its provisions;

        b. You have read and fully understand this Agreement in its entirety;

        c. Your **Build Materials** are either original or do not include any **Software** obtained under a license that conflicts with the obligations contained in this Agreement;

        d. To the best of your knowledge, your **Build Materials** do not infringe or misappropriate the rights of any person or entity; and,

        e. You will regularly monitor the **Website** for any notices.

2. **DEFINITIONS AND INTERPRETATION**

    1. For purposes of this Agreement, certain terms have been defined below and elsewhere in this Agreement to encompass meanings that may differ from, or be in addition to, the normal connotation of the defined word.

        a. "**Additional Code**" means **Software** in source code form which does not contain any

            i. of the **Source Code**, or

            ii. derivative work (such term having the same meaning in this Agreement as under U.S. Copyright Law) of the **Source Code**.

        b. "**AT&T Patent Claims**" means those claims of patents (i) owned by AT&T and (ii) licensable without restriction or obligation, which, absent a license, are necessarily and unavoidably infringed by the use of the functionality of the **Source Code**.

        c. "**Build Materials**" means, with reference to a **Derived Product**, the **Patch** and **Additional Code**, if any, used in the preparation of such **Derived Product**, together with written instructions that describe, in reasonable detail, such preparation.

        d. "**Derived Product**" means a **Software Product** which is a derivative work of the **Source Code**.

        e. "**IPR**" means all rights protectable under intellectual property law anywhere throughout the world, including rights protectable under patent, copyright and trade secret laws, but not trademark rights.

        f. "**Package**" means a computer file containing the exact same contents as the computer file from the **Website** which will be downloaded after accepting, or was opened to access, this Agreement.

        g. "**Patch**" means **Software** for changing all or any portion of the **Source Code**.

        h. "**Proprietary Notice**" means the following statement:
        This product contains certain software code or other information ("AT&T Software") proprietary to AT&T Corp. ("AT&T"). The AT&T Software is provided to you "AS IS". YOU ASSUME TOTAL RESPONSIBILITY AND RISK FOR USE OF THE AT&T SOFTWARE. AT&T DOES NOT MAKE, AND EXPRESSLY DISCLAIMS, ANY EXPRESS OR IMPLIED

WARRANTIES OF ANY KIND WHATSOEVER, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WARRANTIES OF TITLE OR NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS, ANY WARRANTIES ARISING BY USAGE OF TRADE, COURSE OF DEALING OR COURSE OF PERFORMANCE, OR ANY WARRANTY THAT THE AT&T SOFTWARE IS "ERROR FREE" OR WILL MEET YOUR REQUIREMENTS. Unless you accept a license to use the AT&T **Software**, you shall not reverse compile, disassemble or otherwise reverse engineer this product to ascertain the source code for any AT&T **Software**.

i. "**Software**" means, as the context may require, source or object code instructions for controlling the operation of a central processing unit or computer, and computer files containing data or text.

j. "**Software Product**" means a collection of computer files containing **Software** in object code form only, which, taken together, reasonably comprise a product, regardless of whether such product is intended for internal use or commercial exploitation. A single computer file can comprise a **Software Product**.

k. "**Source Code**" means the **Software** contained in compressed form in the **Package**.

l. "**Website**" means the Internet website having the URL `http://www.research.att.com/sw/download/` AT&T may change the content or URL of the **Website**, or remove it from the Internet altogether.

2. By way of clarification only, the terms **Package**, **Proprietary Notice** and **Source Code** when used in this Agreement shall mean the materials and information defined by such terms without any change, enhancement, amendment, alteration or modification (collectively, "changes").

3. **GRANT OF RIGHTS**

1. Subject to third party intellectual property claims, if any, and the terms and conditions of this Agreement, AT&T grants to you under:

a. the **AT&T Patent Claims** and AT&T's copyright rights in the **Source Code**, a non-exclusive, fully paid-up license to:

i. Reproduce and distribute the **Package**;

ii. Display, perform, use, and compile the **Source Code** and execute the resultant binary **Software** on a computer;

iii. Prepare a **Derived Product** solely by compiling **Additional Code**, if any, together with the code resulting from operating a **Patch** on the **Source Code**; and,

iv. Execute on a computer and distribute to others **Derived Products**,

except that, with respect to the **AT&T Patent Claims**, the license rights granted in clauses (iii) and (iv) above shall only extend, and be limited, to that portion of a **Derived Product** which is **Software** compiled from some portion of the **Source Code**; and,

b. AT&T's copyright rights in the **Source Code**, a non-exclusive, fully paid-up license to prepare and distribute **Patches** for the **Source Code**.

2. Subject to the terms and conditions of this Agreement, you may create a hyperlink between an Internet website owned and controlled by you and the **Website**, which hyperlink describes in a fair and good faith manner where the **Package** and **Source Code** may be obtained, provided that, you do not frame the **Website** or otherwise give the false impression that AT&T is somehow associated with, or otherwise endorses or sponsors your website. Any goodwill associated with such hyperlink shall inure to the sole benefit of AT&T. Other than the creation of such hyperlink, nothing in this Agreement shall be construed as conferring upon you any right to use any reference to AT&T, its trade names, trademarks, service marks or any other indicia of origin owned by AT&T, or to indicate that your products or services are in any way sponsored, approved or endorsed by, or affiliated with, AT&T.

3. Except as expressly set forth in Section 3.1 above, no other rights or licenses under any of AT&T's **IPR** are granted or, by implication, estoppel or otherwise, conferred. By way of example only, no rights or licenses under any of AT&T's patents are granted or, by implication, estoppel or otherwise, conferred with respect to any portion of a **Derived Product** which is *not* **Software** compiled from some portion, without change, of the **Source Code**.

4. **YOUR OBLIGATIONS**

1. If you distribute **Build Materials** (including if you are required to do so pursuant to this Agreement), you shall ensure that the recipient enters into and duly accepts an agreement with you which includes the minimum terms set forth in Appendix A, *see the website for this Appendix*, (completed to indicate you as the LICENSOR) and no other provisions which, in AT&T's opinion, conflict with your obligations under, or the intent of, this Agreement. The agreement required under this Section 4.1 may be in electronic form and may be distributed with the **Build Materials** in a form such that the recipient accepts the agreement by using or installing the **Build Materials**. If any **Additional Code** contained in your **Build Materials** includes **Software** you obtained under license, the agreement shall also include complete details concerning the license and any restrictions or obligations associated with such **Software**.
2. If you prepare a **Patch** which you distribute to anyone else you shall:
   a. Contact AT&T, as may be provided on the **Website** or in a text file included with the **Source Code**, and describe for AT&T such **Patch** and provide AT&T with a copy of such **Patch** as directed by AT&T; or,
   b. Where you make your **Patch** generally available on your Internet website, you shall provide AT&T with the URL of your website and hereby grant to AT&T a non-exclusive, fully-paid up right to create a hyperlink between your website and a page associated with the **Website**.
3. If you prepare a **Derived Product**, such product shall conspicuously display to users, and any corresponding documentation and license agreement shall include as a provision, the **Proprietary Notice**.

5. **YOUR GRANT OF RIGHTS TO AT&T**

1. You grant to AT&T under any **IPR** owned or licensable by you which in any way relates to your **Patches**, a non-exclusive, perpetual, worldwide, fully paid-up, unrestricted, irrevocable license, along with the right to sublicense others, to (a) make, have made, use, offer to sell, sell and import any products, services or any combination of products or services, and (b) reproduce, distribute, prepare derivative works based on, perform, display and transmit your **Patches** in any media whether now known or in the future developed.

6. **AS IS CLAUSE / LIMITATION OF LIABILITY**

1. The **Source Code** and **Package** are provided to you "AS IS". YOU ASSUME TOTAL RESPONSIBILITY AND RISK FOR YOUR USE OF THEM INCLUDING THE RISK OF ANY DEFECTS OR INACCURACIES THEREIN. AT&T DOES NOT MAKE, AND EXPRESSLY DISCLAIMS, ANY EXPRESS OR IMPLIED WARRANTIES OF ANY KIND WHATSOEVER, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, WARRANTIES OF TITLE OR NON-INFRINGEMENT OF ANY **IPR** OR TRADEMARK RIGHTS, ANY WARRANTIES ARISING BY USAGE OF TRADE, COURSE OF DEALING OR COURSE OF PERFORMANCE, OR ANY WARRANTY THAT THE **SOURCE CODE** OR **PACKAGE** ARE "ERROR FREE" OR WILL MEET YOUR REQUIREMENTS.
2. IN NO EVENT SHALL AT&T BE LIABLE FOR (a) ANY INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OF OR INABILITY TO USE THE **SOURCE CODE** OR **PACKAGE**, EVEN IF AT&T OR ANY OF ITS AUTHORIZED REPRESENTATIVES

HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, (b) ANY CLAIM ATTRIBUTABLE TO ERRORS, OMISSIONS, OR OTHER INACCURACIES IN THE **SOURCE CODE** OR **PACKAGE**, OR (c) ANY CLAIM BY ANY THIRD PARTY.

3. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. IN THE EVENT THAT APPLICABLE LAW DOES NOT ALLOW THE COMPLETE EXCLUSION OR LIMITATION OF LIABILITY OF CLAIMS AND DAMAGES AS SET FORTH IN THIS AGREEMENT, AT&T'S LIABILITY IS LIMITED TO THE GREATEST EXTENT PERMITTED BY LAW.

7. INDEMNIFICATION

1. You shall indemnify and hold harmless AT&T, its affiliates and authorized representatives against any claims, suits or proceedings asserted or commenced by any third party and arising out of, or relating to, your use of the **Source Code**. This obligation shall include indemnifying against all damages, losses, costs and expenses (including attorneys' fees) incurred by AT&T, its affiliates and authorized representatives as a result of any such claims, suits or proceedings, including any costs or expenses incurred in defending against any such claims, suits, or proceedings.

8. GENERAL

1. You shall not assert against AT&T, its affiliates or authorized representatives any claim for infringement or misappropriation of any **IPR** or trademark rights in any way relating to the **Source Code**, including any such claims relating to any **Patches**.
2. In the event that any provision of this Agreement is deemed illegal or unenforceable, AT&T may, but is not obligated to, post on the **Website** a new version of this Agreement which, in AT&T's opinion, reasonably preserves the intent of this Agreement.
3. Your rights and license (but not any of your obligations) under this Agreement shall terminate automatically in the event that (a) notice of a non-frivolous claim by a third party relating to the **Source Code** or **Package** is posted on the **Website**, (b) you have knowledge of any such claim, (c) any of your representations or warranties in Article 1.0 or Section 8.4 are false or inaccurate, (d) you exceed the rights and license granted to you or (e) you fail to fully comply with any provision of this Agreement. Nothing in this provision shall be construed to restrict you, at your option and subject to applicable law, from replacing the portion of the **Source Code** that is the subject of a claim by a third party with non-infringing code or from independently negotiating for necessary rights from the third party.
4. You acknowledge that the **Source Code** and **Package** may be subject to U.S. export laws and regulations, and, accordingly, you hereby assure AT&T that you will not, directly or indirectly, violate any applicable U.S. laws and regulations.
5. Without limiting any of AT&T's rights under this Agreement or at law or in equity, or otherwise expanding the scope of the license and rights granted hereunder, if you fail to perform any of your obligations under this Agreement with respect to any of your **Patches** or **Derived Products**, or if you do any act which exceeds the scope of the license and rights granted herein, then such **Patches**, **Derived Products** and acts are not licensed or otherwise authorized under this Agreement and such failure shall also be deemed a breach of this Agreement. In addition to all other relief available to it for any breach of your obligations under this Agreement, AT&T shall be entitled to an injunction requiring you to perform such obligations.
6. This Agreement shall be governed by and construed in accordance with the laws of the State of New York, USA, without regard to its conflicts of law rules. This Agreement shall be fairly interpreted in accordance with its terms and without any strict construction in favor of or against either AT&T or you. Any suit or proceeding you bring relating to this Agreement shall be brought and prosecuted only in New York, New York, USA.