SimTK Installation System Requirements

created by Christopher Bruns
August, 2005

Version 1.1

Contents

1 Purpose of this document

This document defines what the SimTK Installation System should be able to do and what its components will be, including both software requirements and a rough specification of the documents needed to allow people to be able to use this system successfully.

This is neither a specification nor a user manual. This is not a user interface design document.

2 Introduction

2.1 What is installation?

Software installation is the process of the delivery of a software product, from a distribution source such as SimTK.org, to a usable form on a user's computer. This delivery can occur in several stages, including packaging, transfer, dependency resolution, building, file installation, update, and uninstall.

2.2 Installation of SimTK software

Part of the mission of SimBios is to create a software tool kit (SimTK) that can be used by biomedical scientists around the world. In order to have the broadest impact, we need to ensure that:

1. It is *easy* and *pleasant* for potential users to find, install, update, and uninstall the SimTK software, and
2. It is obvious to potential users HOW to find, install, update, and uninstall the SimTK software.

These two requirements are often confused, but are not the same thing. This document describes features and usability issues during software installation. It is important to facilitate the entire software installation experience, from the moment the user feels a desire for a software tool that SimTK offers and enters a few terms into Google, all the way through to the time the user has uninstalled the software after the software has completed its mission. Usability issues encountered in the actual use of installed applications and libraries are beyond the scope of this document; though some of the principles are the same.

Requirements for a user-centric software installation system include:

1. Ease of use: one-click installation, or other similarly simple and familiar action. Only involve the user in critical decisions, and cluster those decisions in one brief interval of the installation.
2. Platform auto-detection: Install the right versions of binaries for the item being installed.
3. Recursive dependency installation: Install all dependencies.
4. Painless updating: Keep track of update availability and reinstall if necessary.
5. Good neighbor: Avoid incompatibilities with other software by carefully managing software versions.

The SimTK installer will manage these tasks.

3 Installation functionality for SimTK 1.0

3.1 Issues that need to be resolved

The following issues will need to be answered in time for SimTK 1.0

3.1.1 Can Java Web start be used as a wrapper for non-Java applications?

It might be possible to wrap non-Java applications in java for use in web start. For completely wrapped applications, as might be desired for command-line tools, a Java wrapper would probably need to handles input, output, and error streams. It may be impossible.

Issues:

- Is it possible to run an executable from within a jar file using java's Runtime.getRuntime.exec() command?
- If running such an executable is possible, can that executable gain access to shared libraries in the jar file?

If a Web Start is used as an installer, rather than as a wrapper, it may be necessary for the installed non-Web-Start application to check for updates itself.

3.2 Target computer platforms

Installation of SimTK 1.0 software will be possible on the following platforms:

1. Windows XP

Probably the most populous user class.

2. Linux i386 (32 bit and 64 bit)

Probably the most important platform for distributed computing.

3. Mac OS X

Important class of less technical academic end users.

3.3 Target user classes

3.3.1 Clinical/bench scientists

Scientific users should be assumed to have a low tolerance for user intervention in the install process. One click installation is the goal for these users. Primary installation documentation for these users should consist of "click here to install", with troubleshooting information available on another web page.

3.3.2 Application developers

These users need to be able to deliver applications using the SimTK install process. These developers will need straightforward documentation on packaging software for installation.

3.3.3 Modelers

Modelers are users who will be primarily interested in applications and models.

The structure of the models that may be stored at SimTK.org is not well defined at this point, so it is not possible to cover the installation scenarios for these. For the time being, modelers can retrieve models, in whatever form they are stored, by using direct read access to the project subversion repository. If and when models are stored on SimTK.org, documentation must be provided to aid other modelers in retrieving those models.

Applications directed toward modelers can be delivered by the same means as applications directed at scientists (see above).

3.3.4 Algorithm developers

Algorithm developers will access SimTK software via the project management and subversion code repository interfaces. No special installation tools are specifically required for this class of users.

Algorithm developers will be able to install libraries and applications using the same methods used by other user classes.

3.4 General Installer requirements

The SimTK installer must be able to detect the target platform type. The target platform information that must be gathered includes the operating system type, operating system version, CPU manufacturer, and CPU model.

The installer must find the correct version of binary files for the target platform type.

The installer must be able to construct reasonable default destination directories for all components, taking into consideration the target platform type, current disk space usage, pre-existence of particular directories (such as /usr/local/bin), and the disk write permissions of the installing user.

Two general classes of installations will be supported: applications and libraries.

3.5 Delivery

3.5.1 Delivery by web browser link

In most cases, software delivery should begin with a click on a browser link. The link text should be underlined, to make clear that it is a hyperlink. The link should be accompanied by a "document" icon (image of a page with the corner folded down) to indicate that this is the real download link, not just a link to another page about the download.

During the delivery it is important to provide feedback. The download dialog must give the user some kind of feedback at least every 250 milliseconds. The default behavior of Web Start does NOT provide this! So it may be necessary to create an additional applet that monitors the download progress.

On the other hand, the user must not be forced to watch the download progress. The user should be able to work in another application without interruption until either the download is complete, a fatal error has occurred, or a necessary user confirmation is required.

All user input, including confirmations, configurations, and install locations should be collected in a short interval of time, rather than spread out during the installation process.

3.5.2 Delivery by name

There should be a client-side tool like apt, by means of which a software component and its dependencies can be installed by name. For example "simtkapt get cvode-component".

3.6 Dependency Resolution

Two situations should be addressed separately: If the missing dependency is available from SimTK.org, then we need a uniform mechanism for resolving that dependency. In

the general case where a dependency is from a third party, we need to inform the user, as effectively as possible, of what they need to do to resolve that dependency.

Each software module must keep track of its own dependencies. For each software component that a particular component depends upon, the following should be stored:

1. Name of the dependency
2. Location of the dependency's distribution site
3. Preferred version of the dependency
4. Minimum version of the dependency
5. Maximum version of the dependency ("unlimited" is an acceptable value)

The installer must be able to query these values for a particular component, sense whether the dependency is already installed, and, if necessary, install the missing dependencies.

Implementation detail: For Java web start applications, all dependencies should be specified by referring to jnlp files related to the dependency.

The SimTK Installer must be able to launch separate instances of itself, Java Web Start (JWS), and other installers, to support installation of a broad variety of installation dependencies.

3.7 Build

The core SimTK build process to be used by developers is beyond the scope of this document and will not be discussed here. The SimTK install tool will only be concerned with the installation of non-source code products.

3.8 File Installation:

3.8.1 General installation requirements

The following excerpt from http://mindprod.com/jgloss/installer.html enumerates a good set of installer tool requirements:

1. An install should display a pleasing graphic to start. This gives confidence that the product is of professional quality. However, it is quite unnecessary to have a Star-Wars level animated production to entertain during the setup. There should be no need to even look at the install except the first screen on two.
2. The install should ask all its questions **up front** so you can have a short conversation with it, then **walk away** or run it in the background. If it needs a blank diskette, it should ask for it up front. If it wants a branding key, it should ask up front, not after 40 minutes.
3. The install should check for free space on local hard disks, and display them in descending order. By default, it should select the one with the most free space, not necessarily C:. Selecting the drive should be a separate question from naming the directory.

4. If an install required downloading 10 files, you should select the options up front, then go home and leave the install to automatically download all the files overnight and install them unattended.
5. The install should ask only one question per dialog box. It is too easy for the user to miss one of the questions otherwise, especially the crucial one, which drive to install on.
6. The install should be prepared to run in the background unless it is a system level install. Programmers with immense egos believe that users have nothing better to do that watch their installs copy files. Their installs typically grab the entire screen and insist users watch them to completion, making users hit Enter every 15 seconds just to ensure they savor every little animated burble of ad copy.
7. Installs should not require rebooting the machine, especially not more than once, even for system software. [In particular, the SimTK installer should make a thorough inventory of in-use DLLs that must be replaced, and replace them all upon a single reboot].
8. The install should display a conservative countdown timer of how much longer this is going take.
9. Upgrade installs should carefully preserve existing preferences, configurations, plug-ins, hotlists, scripts etc. converting to new formats if necessary.
10. Where it makes sense, upgrades should offer you the choice of preserving the old version. Where it does not make sense, the install should leave a result indistinguishable from a clean install of the new version and not leave junk files and registry entries left behind from the old version. It is extremely common for installs to fail over a previous version or a previously aborted install. Vendors should test the install to make sure it does not choke over any junk left behind by previous attempts.
11. Java Web Start is the easiest way to implement one-click installs, and to ensure automatic updating. Java Web Start is rapidly becoming **the** way all Java applications are delivered. It is so much more convenient for the end user. It is free. The downloads are tiny. It is very little work to prepare a Web Start app. The only serious drawback is to work properly it needs a non-standard file server to deal with versioning.
12. Check your questions with a variety of people to see if *anyone* can misinterpret them in an ambiguous way. One I question I saw recently "Exiting now will have a horrible consequence. Continue?" Did they mean continue with the exit or continue running the program without exiting? I often write letters to such authors telling them of the ambiguity. The fools write back telling me which they intended, and explain why there is no problem with ambiguity if you look at it the *right* way.

[end of excerpt from http://mindprod.com/jgloss/installer.html]

In cases where system libraries or other system resources might be replaced by an application, the SimTK installer should be able to put all of its libraries and resources into a private sandbox area, for the exclusive use of the installed software product, and related products from SimTK.org.

3.8.2 License and registration

The number of responses required from the user should be kept to a minimum. On the other hand, there can be context that is important to present to the user.

3.8.2.1 Software License

Either the text of the software license itself, or a brief description and link to the license text should be displayed prior to installing software files.

Two modes of interaction will be possible, as a configurable option to the application designer: 1) The license or description appears for a finite period of time with no response required from the user; or 2) The user must click either “Agree” or “Disagree”. After clicking “Disagree”, the installation will be aborted and cleaned up.

For software products that include multiple copyrighted works, a summary dialog will display each of the copyright holders with a link to the copyright text.

3.8.2.2 Optional Registration

Some software product administrators may want to require users to register before installing software. In these cases, the user will need to complete a registration form before installing the software. The registration dialog must mention that not all SimTK software requires registration. The registration dialog must make clear which data fields are required, and which are optional. The registration form must include a privacy statement, indicating how the data will and will not be used. The data that are collected must be stored in a reasonably secure manner. The set of individuals who have access to the registration data must be tightly controlled.

3.8.2.3 Scholarly publications

When the software product is associated with scholarly publications, a non-modal dialog with the full citations, as they might appear in the references section of a journal article, should be shown. These publications should also be viewable from the “about” dialog of software applications.

3.8.3 Installation of application files

Different installation issues apply to applications, libraries, command-line tools, and associated data files and documents.

3.8.3.1 End-user GUI applications

Application binaries must be delivered to the proper system location, which will depend upon the system type.

In the case where a smooth installation method is not provided for a software product, there should at least be clear step by step instructions on how to build and install the product from source code.

3.8.3.2 End-user command-line applications:

Command-line applications require no graphical user interface, and thus some users may not use a graphical user interface. This is one case where requiring a click to install could be a burden. Even for utilities distributed as binaries:

If the command-line tool depends only upon standard libraries, it should be delivered as an uncompressed executable file, perhaps with on-line instructions of where to put it.

If the tool depends upon non-standard libraries, the installation should use the same installation method as graphical applications.

3.8.3.3 Data files and screen icons

Icon files and other application-specific files must be delivered to the proper location, which will, in general, depend upon the target platform type.

3.8.3.3.1 Java Web Start applications

For applications delivered using Java Web Start, auxiliary files should be delivered in jar files, and accessed from within the Java program using the ClassLoader.getResource("<filename>") mechanism.

3.8.3.3.2 Macintosh application bundles

Macintosh application bundles should include auxiliary files to be used by the application in the resource fork of the application.

3.8.3.3.3 Other installation types:

For Windows applications, application specific auxiliary files should be placed in the folder associated with the application, such as "C:\Program Files\CoolProgram\Resources\".

3.8.4 Installation of library files

Binary libraries must be delivered to the proper system library location, which will depend upon the target platform type.

3.8.4.1 Precompiled shared libraries

Installation of libraries for system-wide use should be placed in system areas. The installation must test for the presence and version of existing libraries. If preexisting libraries are replaced, backup copies of the old libraries must be created. And a method for backing out the installation should be provided.

It should be possible for the installing user to override the default library installation location, but it should also be very easy for the user to simply accept the default location.

For example, on Linux or other UNIX-like operating system, the installation location for shared libraries should default to directory such that:

1. Directory is contained in LD_LIBARARY_PATH environment variable

2. Directory exists
3. Directory is writable by the current user
4. Directory is one of:
    a. /usr/local/lib
    b. /usr/lib
    c. ${HOME}/lib

If no directory satisfies these criteria, the user should be prompted to choose an installation directory from LD_LIBRARY_PATH, or choose a new directory by browsing. If the user chooses a directory not on LD_LIBRARY_PATH, the user should be given helpful encouragement to modify the value of LD_LIBRARY_PATH.

In cases where the installing user does not have admin rights, and does not have write permission to locations such as /usr/local/bin, the installer must communicate in brief, clear language that the installation will not be system-wide; that the installation will be for that user only; and that someone with admin privileges would be required to install the software system-wide.

Similar requirements should be used for installing executable binaries, but replacing “lib” with “bin”, and replacing “LD_LIBRARY_PATH” with “PATH”.

In case the replacement of in-use system libraries will cause trouble, the SimTK installer must prepare for at most one reboot cycle to replace these libraries.

The installation of auxiliary files and documents will also depend upon the target system type.

3.8.4.2 Precompiled static libraries

Same process as that for shared libraries.

3.8.5 Installation of document files

User documents and tutorials should be provided in HTML format exactly one click away from the project main page at simtk.org. Applications should provide a direct link to these documents. In any installation where there is an installation directory under the control of the installing user, a set of documents should be installed there too.

3.8.6 Security

Delivered software components must be signed using trusted certificates. The installing user will need to accept the certificate to install the software. The total number of certificates should be kept small for each software component. On the other hand, the set of certificates should show an accurate view of the sources of the software components being installed.

3.8.7 Setting system variables

The SimTK installer must be able to modify the setting of environment/registry variables for use by applications and libraries. This should include the ability to add to library and

executable paths, though the use of preexisting system areas for applications and libraries should be encouraged.

3.9 Configuration

Keep configuration options to a minimum. Only ask questions that it is truly important to the user to answer. Do not rely on the user to make decisions that the developer is too lazy to make.

3.10 Running

The details of running behavior for installed applications are beyond the scope of this document. However, it is required that runnable applications are easily run, after the installation and configuration are completed.

3.11 Updating

The first step of potential software updating is to compare the software components on the user's computer with recent versions on the distribution server. This comparison process must be able to answer the following questions:

1. Is the user's software identical to the recommended latest version on the server?
2. Is the user's software a newer or older version than that stored on the server?
3. Is the user's software a version that was distributed by the distribution server, or is it an "unofficial" variant?
4. What is the stability category of the user's software? (production stable, interim release semi-stable, bleeding edge development version, custom branch version, third party branch version)
5. Finally, is there a newer version of the software on the server that
    a. In the case of libraries, maintains a compatible API with the version on the server?
    b. In the case of applications, maintains compatibility with the local user data that a user might have generated?
    c. These conditions will be difficult to enforce, but need to be considered. Those libraries that do attempt to maintain this kind of compatibility should be able to advertise themselves as such.

To accomplish these tasks, each software component must be able, at a minimum, to know the address of the main distribution server, and to communicate identifying information about itself to the server.

Each application should be able to check for updates to itself each time it is started, and at the user's request. Failure to connect to the update server must be handled quietly and gracefully.

3.12 Uninstalling

Each software component must be able to be uninstalled, restoring the system, insofar as possible, to the state prior to the install. If libraries have been installed in system areas for the application, these should be treated carefully. In the lamest case, the user should

be warned about the library removal. Ideally, the uninstall process should be savvy about which libraries are likely to be used by other applications.

3.13 Language versions

Some SimTK software may require particular versions of software languages. The SimTK installer must be sensitive to this.

3.13.1 C/C++

Requirements for a particular version of C or C++ are handled by the build environment, and are beyond the scope of this document, and beyond the scope of the SimTK installer.

3.13.2 Java/Perl/Python/Tcl

The SimTK installer must be able to detect installed versions of these languages on the client machine, and install a different version if that is required. Such a significant install should be done in a private sandbox area rather than system wide.

If the user wishes to perform a system scale installation of these tools, the SimTK Installer must guide them to the primary distribution sites for these tools.

4 What will not be in SimTK 1.0

4.1 Dependency resolution for non SimTK software

One exception to this is dependency resolution for Java Web Start applications, which will be handled by defining external resources in the jnlp files.

4.2 Installation from source code

User level installation from source code will not be included in the SimTK installer.

5 Documentation needed

The following documents will be needed to standardize the software delivery process. This list does not include the software-specific documentation that is needed for each software product.

1. Guide to deployment using Java web start (for application developers)
2. Guide to platform-specific installers (for application developers)
3. General guide to installing SimTK software (for users)

6 References

https://simtk.org/home/planning/
http://mindprod.com/jgloss/installer.html

7 Acknowledgments

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research Grant U54 GM072970.

Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

8 Appendix A: Overview of existing installation tools

Software installation generally consists of the following:

1. Packaging
2. Delivery/Acquisition
3. Dependency resolution
4. Building
5. File Installation
6. Configuration
7. Running
8. Updating

Not all of these steps are necessary in every case, and some steps may be more or less transparent to the user. The degree of automation that is possible in each of these steps depends upon several factors, including the target user class, the type of binaries/sources that are provided, and the target computer platform.

8.1 Packaging

Packaging is the method by which the software creator encapsulates the software into a form that can be delivered to the user. The following types of packaged software are common:

1. Self-extracting binaries. These binaries typically contain both installation software and the end-user software. Such binaries can be created using commercial software such as:
    a. Mac: PackageMaker
    b. Windows: iexpress, InstallShield, InstallAnywhere, WinInstall, and PackageStudio
    c. Linux: self extracting shell archives
2. Stand-alone binaries.
3. Executable Java jar files
4. Macintosh application bundles. These look like single files, but secretly contain complex structure.
5. Well known archive formats, such as .tar.gz, rpm, binhex, or .zip. The installation process following download can range from a) the user manually unpacking the archive and then following build instructions, to b) automated installation, beginning with automatic recognition by the user's system of the archive format.
6. Java jar files with jnlp file meta-information, for use with Java Web Start.
7. Remotely accessible CVS or SubVersion repositories are also a kind of software packaging.

8.2 Acquisition/Delivery

Acquisition (or delivery, from the standpoint of the software maker) is the process by which the user's computer receives the initial software package. Traditional means by which this is accomplished include:

1. Plug-in based acquisition, as with Java Web Start. With Web Start, many aspects of installation can be handled, but the user must first have installed a recent Java run time environment (JRE 1.3 or later). See appendix for details.
2. Extraction of software versions from a CVS or SubVersion revision history archive. Such installation customarily requires a build step, but it is possible to deliver binary distributions in this way as well.
3. Most delivery formats can be obtained from ftp sites, http web sites, or physical installation media such as CD's.

8.3 Dependency resolution

Many software components will not work unless other software components are also installed. In the general case, an acyclic directed graph of software dependencies requires a series of installation steps before the target software component can finally be installed. Some approaches to dependency resolution include:

1. Expert user resolution: Specify the dependencies in the install documentation, prompting the user to fetch the dependencies. The user, upon examining the documentation for the dependencies, may then discover a third layer of software that needs to be installed, and so on. The disadvantage of this approach is that traversal of the DAG installation structure is an exercise left to the end user.
2. Tools which recursively install dependencies. This usually requires that the software be packaged in a uniform way, using machine-readable dependency documentation. Examples of this approach include:
    a. apt-get
    b. rpm
    c. perl cpan
    d. yum
    e. Java web start (dependencies must be explicitly declared in jnlp files)

For Java web start applications, all dependencies should be specified by referring to jnlp files related to the dependency. This needs to be done manually.

8.4 Building

A full discussion of build processes is beyond the scope of this document. See the SimTK build environment requirements document for details.

The most familiar tool for building is the make utility. When a makefile is present, the command "make install" should install the software on the user's computer.

8.5 File Installation

The method of installation depends upon the type of software being installed, and the delivery medium.

8.5.1 Archived formats (tar, binhex, zip)

The installation process following download can range from a) the user manually unpacking the archive and then following build instructions, to b) automated installation, beginning with automatic recognition by the user's system of the archive format.

8.5.2 Installation of binary files

8.5.2.1 Java Web Start applications

Binaries are maintained automatically in Web Start's disk cache.

8.5.2.2 Other installation situations

Custom installer typically installs binaries in various directories, under the guidance of the installing user.

8.5.3 Installation from source

Installation from source is beyond the scope of this document, which focuses on end-user installation issues. See the build environment requirements for details.

8.6 Configuration

In addition to installation directories, there may be other options that the user should set during installation.

8.7 Running

For end-user applications, the run process should be initiated using familiar methods: desktop icon, start menu, or command line execution using the application name plus optional arguments.

In the case of shared libraries, which are not executed directly, it suffices for the libraries to be located where they can be found automatically by dependent applications.

In the case of static libraries, they should be located where they can be found by the linker at build time for dependent applications.

8.8 Updating

Updating an existing installation with new binaries can be tricky.

Some existing tools for managing updates include:
apt-get/rpm/yum
Java Web Start (JWS)

8.9 Uninstalling

Most applications that have been installed using a professional installer have a straightforward uninstall process.

8.10 One Unified Solution: Java Web Start

With the exception of the build process, Java Web Start handles all of the phases of software installation (except packaging) for Java applications.
Advantages to web start:

- Fully automated, Web-centric distribution and installation of Java 2 applications, applets, and extensions based on the JNLP;
- Resource caching—Application components are cached automatically on the client's machine;
- Browser independence—Applications are executed outside of the browser process and can also be launched directly from the desktop;
- JVM independence—A pre-requisite virtual machine implementation and version can be specified and, if not already present on the client machine, downloaded and installed automatically;
- Transparent updating—Versions of cached application resources are checked against those hosted on the Web server. Newer versions are downloaded and installed automatically;
- Incremental updates—Only new or modified classes and resources need be uploaded to the client's machine; and
- Incremental downloads—if required, archives can be downloaded only when first required as opposed to being downloaded immediately.
- Support for multiple JVMS
- jar file versioning, which allows users to update an application without downloading the entire application
- The developer can choose whether the user can download an application to run it later when not connected to the network.
- Support for applets, applications in an applet like sandbox and full applications.
- Uninstall of applications is accomplished relatively painlessly through the web start application itself.

Problems with web start:

- Cannot run Java 1.5 applications on the Mac
- Cannot run Java 1.5 applications on 64-bit Linux
- Might only work with Java applications
- Limited feedback, no built-in size-check as far as I can tell (all these need to be added manually and placed in a common package to be used by all JWS installed software)
- Bugs (e.g. dependent resources loading which may be problematic for circular dependencies).

- Potential instability when a new JRE is introduced by SUN (we may have to certify some JREs)