www.A-A-P.org

user

install

When an end user desires to install an application many questions arise:
1. Where to download it from?
2. Is this version going to work on my system?
3. Are there any fixes I should include?
4. What libraries and tools does this application require?

A-A-P makes this easy: The end user only needs to find the top level recipe for the application (see below). It contains all the information required to install the application:

1. A list of sites where the application can be downloaded from. Possibly different sites for different systems.
2. The right version of the application will be selected, based on the requirements of the application, desired version (stable or latest) and properties of the system (OS version, available libraries).
3. When installing from source code (like the FreeBSD ports) patches are located and included when needed. For binaries patches may also be applied.
4. Dependencies on other packages are handled. A range of acceptable versions can be specified. Obtaining several standard tools is handled by A-A-P itself (e.g., unzip).

Delayed installation: From within an installed application the recipe may be used again to install an extra feature. This is especially useful for documentation in various languages. It avoids that many choices need to be made when installing the application.

Updating of an existing application is supported. It can first be installed to try it out and delete the previous version only when there is no problem.
A-A-P will retrieve those files that have changes. The developer still has to provide the logic how settings are taken over by the new version. Python script can be used for this.

why not use a Makefile?

Portability:
Makefiles are not portable. Not only because there are several variants of the make program (e.g., GNU make and BSD make), also because they use shell commands that may not be available everywhere (use "copy" or "cp"?) or have subtle differences (e.g., options for "cp").
A-A-P recipes use Python commands, which execute the same way on all systems. The intention is that A-A-P works at least on all kinds of Unix, MS-Windows and MacOS.

Reliability:
Makefiles only use timestamps for dependencies. In many situations this fails (e.g., after restoring an older version of a file). A "make clean" or "touch" is often needed to work around this.
A-A-P uses various signatures. For C files the signature ignores changes in comments and irrelevant white space changes. For most other files MD5 checksums are used. Timestamps are also available.

Power:
Makefiles are very simplistic and contain redundant information, which is often solved by using tools to generate or update the Makefile. The lack of a script language results in using shell commands and other clumsy solutions. A-A-P is much more intelligent. The default rules take care of most of the common things (e.g., updating dependencies, compiling for various languages). Python script can be used for the complicated things.

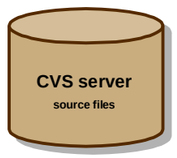
search

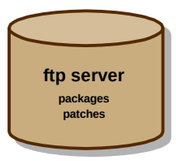
Finding the top level recipe is still required to install an application. Several ways to locate the recipe are provided:
- The central recipe repository can be searched. Everybody can upload a recipe here. A rating system helps to select between alternatives.
- The site for the application provides the recipe.
- The site for your OS provides recipes for ported applications.
- Use a search engine.
- A collection of recipes is already on your system (similar to the FreeBSD ports system). A-A-P will automatically find an updated version.

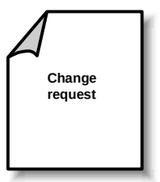
If no recipe can be found, the user can write it himself. Examples and guidelines are provided to make this easy. This is a bit more work than manually obtaining the files. The advantage comes when installing a new version or installing the same application on another system. And the recipe can be made available to others, so that they don't have to reinvent the wheel.

Current status of the A-A-P project

The A-A-P project has started March 2002.
No code is available yet!
The first development version is expected by September.
The first release should be available spring 2003.
However, this very much depends on volunteers to join the project.







developer

distribute

The developer writes a recipe with which an end user can install his package.

A common method is to create a simple top level recipe, which detects the type of system and refers to an OS-specific recipe. Python commands can be used to detect properties of the system.

A choice can be made to do the whole installation with recipes or to use an OS-specific installer once the required files have been downloaded.

The actual distribution (uploading) of files can done by the build recipe (see below). This avoids mistakes when making a new release. A minimal distribution has one recipe and an archive with files. A large distribution has several recipes, archives and patches. Files are usually uploaded with ftp, rcp or using cvs. Many other methods are supported.

Useful recipe items:
MASTER_SITE: Specify a list of sites where the files can be obtained. Can also be a standard list (e.g., the FreeBSD sites or sourceforge).
ORIGIN: Where this recipe itself can be obtained. A-A-P uses this to check if an updated version of the recipe is available.
:child refers to a recipe that is used only when needed. It works like including that recipe at this spot, but takes care of changing directory and local variables.

What is a recipe?

A recipe is a mix of a Makefile and a Python script.

Like a Makefile it defines dependencies and actions to carry out when a target needs to be updated.

Python commands can be used almost everywhere. They are in lines marked with @ or are enclosed in **[ ]**.

Many default rules exist to keep the recipe simple and portable. The result is that setting a few specific variables like **SOURCE** and **TARGET** is enough for A-A-P to know what needs to be done.

Special A-A-P commands start with a colon. This makes them easy to recognize and allows for future expansion. Examples are **:rule**, **:update** and **:child**.

develop

A-A-P includes many features to make developing software easier:

The recipe can be used as a powerful Makefile for **building**. At the same time it can be used as a project file for the **IDE**.

The A-A-P tools form a framework. Interfaces are defined to be able to plug-in your favorite choice of tools.

The **browse** system is used to navigate through all the parts of a project. Not only to find where a variable is used, also to locate documentation on the internet.

For large projects A-A-P provides the possibility to only have those parts local that you are working on. The rest is **downloaded on demand**. For example, when jumping to the definition of a function, a browse file tells A-A-P in which file a function is defined and the recipe specifies the URL of the directory where the files can be obtained from.

Uniform support for various **Version Control** Systems (VCS). For the most often used commands no knowledge of the specific VCS tools is required. A personal VCS is provided. It works like an unlimited undo operation with branches.

Automatic **configuration**. Like autoconf, but easier to use and also works on non-Unix systems. Uses Python, just like recipes. (probably not available in the first version.)

Handling of change requests. This interfaces with an existing **issue tracking** system to store and retrieve the reports. The advantage of using A-A-P for this is that many fields can be filled in automatically.

Comments?

Write your comment here.
Please add your name and e-mail address.

A-A-P is funded by NLnet Stichting

free info