ABout Time Linux
(v0.6)

Eric D. Schabell
ericks@abtlinux.org

November 1, 2005

Abstract

This document will detail a project that has risen from the desire to create a generic framework for managing the software on a Linux system. It is based on my experiences while working as a developer on a source-based Linux distribution spanning more than three years at the time of this writing.

I hope to take my experiences as a developer and package maintainer to create this new package manager. I follow in the footsteps of a few pretty good existing projects such as (micropkg, 2004), (Easinstaller, 2004) and (SMGL, 2004), though I have found these to be lacking in some way or another.

1 Introduction

The *ABout Time Linux (AbTLinux)* package managers design will be dealt with in this document. I will cover them in the following sections:

1. **Use Cases** - clearly defined requirements.
2. **Scenarios** - use cases worked out in clear examples.
3. **Design** - details of *AbTLinux* design, worked out in *UML* diagrams.

The driving force behind this project is to design a basic software package management system that will provide the needed infrastructure to not only install software packages, but also be able to maintain the systems it is installing the software packages onto. I believe this can be done using source based packages and by avoiding the existing RPM and DEB package management systems. This is not new, nor do I pretend to have found the Holy Grail of package management tools. The primary goal is a well documented design, with clearly documented coding practices which will result in an easily maintainable package manager. This in turn will be the foundation of this Linux distribution. Make no mistake about it, the clarity of design and coding practices will take the foremost priority in this project.

I will present my ideas in the following sections starting with the use cases that will be used to detail the requirements. These will be followed by scenarios which provide explicit examples for each requirement. Furthermore, I

will put forth a design for a package management system, which will include implementation choices such as the language and system requirements.

2 Use Cases

The requirements for this project are to be defined though *Use Cases* and are the basis for an initial 1.0 release. I wish for the final set of requirements to be those needed to define a basic framework to manage the software on a Linux machine. I consider these to provide that basic functionality:

2.1 General

This section details items that are global in nature the *AbTLinux*:

- source-based distro, binary as a bonus.
- provide configuration file tools (view, edit, diff for smooth package updates).
- suggest to install package if not installed and a command is entered from that package.
- the abt package manager will be a command line interface.

2.2 Packages

This section details the requirements related to the packages themselves:

- install package
  - if new install, just do it and generate logs.
  - if upgrade, install over, compare old/new logs to delete what is not new.
- reinstall a package, either from cached build or rebuild/reconfigure.
- remove package (includes check for lone package dependencies, ask user if wants to remove them)
- downgrade a package to previous version
- freeze a package in its current state (version holding)

2.2.1 Install package

Use Case Name:	**Install package**
Description:	The basic steps to be taken to install a software package.
Actors:	User
Preconditions:	Package description is available.
Triggers:	abt install <pkg>
Basic Course of Events:	1. User submits an install package request. 2. Package to be installed is placed in install queue. 3. *Details* section of package is processed. 4. *Pre* section of package is processed. 5. *Configure* section of package is processed. 6. Copy of configuration is saved. 7. *Prebuild* section of package is processed. 8. *Build* section of package is processed. 9. Copy of build is saved. 10. *Preinstall* section of package is processed. 11. *Install* section of package is processed. 12. A list of installed files is saved. 13. An integrity check is created on each installed file and saved. 14. *Post* section of package is processed. 15. Package is added to installed packages list. 16. Package is removed from install queue. 17. Package build is cached for future usage. 18. Package sources (build directories) are cleaned up (removed). 19. User is notified that package was successfully installed. 20. Check for possible dependency breakage.

Exceptions:	1. The package is already installed, report this and terminate with success. 2. The *details/pre/prebuild/preinstall/post* sections fail, report error and process stops. 3. *Configure* section fails, report error, save copy of configuration and process stops. 4. *Build* section fails, report error, save copy of build log and process stops. 5. *Install* section fails, report error, clean up any installed files (all or nothing is installed) and process stops. 6. Dependency breakage detected, add broken package to install queue. 7. Existing configuration files detected during installation of new files, query user to either; backup old versions and replace with new, keep old in place and copy new one next to it, replace old with new, default to the first choice.
Postconditions:	1. Package is installed. 2. For existing packages, when error is reported, then package is still in install queue. 3. For broken dependencies, package added to install queue.

2.2.2 Reinstall package

Use Case Name:	**Reinstall package**
Description:	The basic steps to be taken to reinstall (from cache or rebuild) a software package.
Actors:	User
Preconditions:	Package description is available and installed on system.
Triggers:	abt reinstall <pkg>
Basic Course of Events:	1. User submits a reinstall package request. 2. User asked if cached version should be installed or rebuild package. 3. Cached version can be installed. 4. Rebuild requested, package added to install queue. 5. Previous configuration details available, query for reconfigure. 6. *Details* section of package is processed. 7. *Pre* section of package processed, reconfigure only if requested. 8. *Configure* section of package processed. 9. Copy of configuration created in temporary file. 10. *Prebuild* section of package processed. 11. *Build* section of package is processed. 12. Copy of build is created in temporary file. 13. *Preinstall* section of package processed. 14. *Install* section of package is processed. 15. A list of installed files is created in temporary file. 16. An integrity check created in temporary file on each installed file. 17. *Post* section of package is processed. 18. Configuration, build, install and integrity temporary files replace originals. 19. Package listing in installed packages list updated. 20. Package removed from install queue. 21. Package sources (build directories) are cleaned up (removed). 22. User notified that package was successfully reinstalled. 23. Check for possible dependency breakage.

Exceptions:	1. Any failures will result in created files (configuration and/or build) being saved with timestamp in file name. 2. Any previous timestamped (unsuccessful reinstalls) will be removed, but original successful installs will remain. 3. Any failures will result in user being given an error message with failure location information. 4. Dependency breakage detected, add broken package to install queue. 5. Existing configuration files detected during installation of new files, query user to either; backup old versions and replace with new, keep old in place and copy new one next to it, replace old with new, default to the first choice.
Postconditions:	1. Package is reinstalled from cached version or rebuilt. 2. New reinstall configuration, build, install and integrity files replace original installed files. 3. For rebuild reinstalls (not for cached reinstalls) that fail, the package remains in the install queue. 4. For broken dependencies, package added to install queue.

2.2.3 Remove package

Use Case Name:	**Remove package**
Description:	Remove an installed package.
Actors:	User
Preconditions:	Requested package is installed.
Triggers:	abt remove <pkg>
Basic Course of Events:	1. User submits a remove package request. 2. Check for dependency breakage, report for user input. 3. Package is added to uninstall queue. 4. Tarball is created containing the following: (a) Copy of installed configuration files. (b) Copy of configuration. (c) Copy of build. (d) List of installed files. (e) List of integrity checks for installed files. (f) Copy of installed files. (g) Copy of source files for this version. (h) Copy of package data (detail, pre, configure, etc). 5. Installed package files are removed. 6. Package is removed from installed packages list. 7. Package is removed from uninstall queue. 8. Check for dependency breakage. 9. User is notified that package was successfully uninstalled.
Exceptions:	If package not installed, exit with success.
Postconditions:	1. The package is no longer installed. 2. There is a tarball containing everything needed to re-create the package that was just removed. 3. For broken dependencies, package added to install or uninstall queues.

2.2.4 Downgrade package

Use Case Name:	**Downgrade a package**
Description:	Downgrade an installed package to a previously built version.
Actors:	User
Preconditions:	Previously requested version must be available.
Triggers:	abt downgrade <pkg> [version]
Basic Course of Events:	1. User submits a request to downgrade a package. 2. Check for dependency breakage, report for user input. 3. If User requests specific version and it is available we run *reinstall use case* for the cached version. 4. if no version selected or version not available, User presented with a list of previously cached versions. 5. User selects version to install. 6. Using package versions tarball (contains all data saved on *remove package use case*), User is led through the *reinstall package use case*. 7. Check for dependency breakage. 8. User is notified that package was successfully downgraded to selected version.
Exceptions:	Requested versions are not available (empty list of cached versions), exit with error message.
Postconditions:	1. Requested package version has been installed, downgrading installed package version. 2. For broken dependencies, package added to install or uninstall queues.

2.2.5 Freeze package

Use Case Name:	**Freeze a package**
Description:	Freeze a package at the currently installed version to prevent updates.
Actors:	User
Preconditions:	Package must be installed.
Triggers:	abt freeze <package>
Basic Course of Events:	1. User requests a package be frozen at current version number. 2. User given chance to add a text note to frozen status. 3. Package install list is annotated to show frozen status. 4. User is notified that package has been frozen.
Exceptions:	1. If package not installed then offer to install and freeze package. 2. If package already has status frozen, complete process as if reset to frozen.
Postconditions:	1. Package has been marked with status frozen. 2. Package will not be queried for possible upgrade (version increase)

2.3 Queries

This section details items related to queries a user / administrator would need to be able to make on *AbTLinux* machines:

- show package details
- show installed packages
- show package install
- show package build
- show frozen packages
- show package dependencies
- show untracked files on system
- show journal
- show package that a file comes from
- search package description
- show install queue (all builds will be via a queue)
- show available update patch list (before retrieving them)

2.3.1 Show package details

Use Case Name:	**Show package details**
Description:	Display the given packages details.
Actors:	User
Preconditions:	Package tree is available.
Triggers:	abt show-details <pkg>
Basic Course of Events:	1. User requests a show package. 2. Package details displayed: (a) name (b) version (c) author (d) source location (e) last package update (f) depencencies (g) description (h) installed version (i) available version (from abt repository)
Exceptions:	none.
Postconditions:	User has seen the given packages details.

2.3.2 Show installed packages

Use Case Name:	**Show installed packages**
Description:	Display a formatted listing of the installed packages.
Actors:	User
Preconditions:	Package install listing and network is available.
Triggers:	abt show-installed
Basic Course of Events:	1. User requests an installed packages report. 2. Access the installed packages listing. 3. Access the remote update listing for most recent package versions. 4. List presented containing the installed package details: (a) name (b) date (c) installed version (d) available version local (e) available version abt repository
Exceptions:	Remote package update listing is not available, complete report by filling in nothing for available version.
Postconditions:	User has been presented an overview of the installed packages.

2.3.3 Show package install

Use Case Name:	**Show package install**
Description:	Display a listing of the files installed for given package.
Actors:	User
Preconditions:	Package requested is installed.
Triggers:	abt show-files <pkg>
Basic Course of Events:	1. User requests to view a listing of the files installed by a package. 2. A dump is given of the installed files log for the package.
Exceptions:	Missing file log, report error.
Postconditions:	User has been presented a listing of the files installed by a package.

2.3.4 Show package build

Use Case Name:	**Show package build**
Description:	Display the build log for given package.
Actors:	User
Preconditions:	Package requested has been built previously.
Triggers:	abt show-build <pkg>
Basic Course of Events:	1. User requests to view the most recent build log for package. 2. A dump is given of the most recent build log for the package.
Exceptions:	Missing build log, report error.
Postconditions:	User has been presented a log of the given packages build.

2.3.5 Show frozen packages

Use Case Name:	**Show frozen packages**
Description:	Display all packages with a frozen status.
Actors:	User
Preconditions:	none.
Triggers:	abt show-frozen
Basic Course of Events:	1. User requests a list of frozen packages. 2. The installed package list is used to determine which packages need to be checked. 3. Packages that are determined to be frozen are displayed.
Exceptions:	Missing install log, report error.
Postconditions:	User has been presented a list of frozen packages.

2.3.6 Show packages dependencies

Use Case Name:	**Show package dependencies**
Description:	Display the dependency tree for a given package.
Actors:	User
Preconditions:	none.
Triggers:	abt show-depends <pkg>
Basic Course of Events:	1. User requests to view dependencies for package. 2. The dependency engine is used to generate a dependency tree (entire tree). 3. The dependency tree is displayed for user to browse (both up and down the tree).
Exceptions:	none.
Postconditions:	User has been presented a dependency tree of the given package.

2.3.7 Show untracked files

Use Case Name:	**Show untracked files**
Description:	Finds and displays a list of files not being tracked by abt.
Actors:	User
Preconditions:	none.
Triggers:	abt show-untracked
Basic Course of Events:	1. User requests to view a list of untracked files. 2. Using install logs as cross reference, list of untracked files is created. 3. Display list of untracked files.
Exceptions:	Missing installed packages or install log files, report error.
Postconditions:	User has been presented a list of untracked files on the system.

2.3.8 Show journal

Use Case Name:	**Show journal**
Description:	Display abt journal.
Actors:	User
Preconditions:	Abt journal exists.
Triggers:	abt show-journal
Basic Course of Events:	1. User requests to view the abt journal. 2. The abt journal is displayed.
Exceptions:	Missing journal, report error.
Postconditions:	User has been presented the journal.

2.3.9 Show file owner

Use Case Name:	**Show file owner**
Description:	Display the owning package for a given file.
Actors:	User
Preconditions:	Package owning file is installed correctly.
Triggers:	abt show-owner <file>
Basic Course of Events:	1. User requests to view the owning package for a given file. 2. The file may given as relative path or complete path. 3. The install logs are searched to determine which package is owner. 4. Owning package is displayed to user.
Exceptions:	File not found to be owned by a tracked package, show warning.
Postconditions:	User has been presented the name of the owning package for given file.

2.3.10 Search package descriptions

Use Case Name:	**Search package descriptions**
Description:	Search all package descriptions for the given string, return package names that match.
Actors:	User
Preconditions:	Search string is normal string or a regexp.
Triggers:	abt search <string or regexp>
Basic Course of Events:	1. User submits a search string. 2. All package descriptions are searched for match to given string. 3. Packages matching are displayed to user.
Exceptions:	No matching string found, return warning.
Postconditions:	User has been presented with a list of packages with string matches from their descriptions.

2.3.11 Show install queue

Use Case Name:	**Show install queue**
Description:	Display the install queue contents.
Actors:	User
Preconditions:	none.
Triggers:	abt show-iqueue
Basic Course of Events:	1. User requests to view the contents of the install queue. 2. The contents of the install queue is displayed.
Exceptions:	Missing install queue, create install queue and report empty queue.
Postconditions:	User has been presented with the contents of the install queue.

2.3.12 Show available patches

Use Case Name:	**Show available patches**
Description:	Display all available patches.
Actors:	User
Preconditions:	Network available.
Triggers:	abt show-patches
Basic Course of Events:	1. User requests to view all available patches. 2. The list of available patches is retrieved from network. 3. List of available patches displayed.
Exceptions:	Retrieval of patch listing fails, report error.
Postconditions:	User has been presented a complete list of available patches.

2.4 Generation

This section details items that need to be generated from an installed machine:

- list of packages needing updates
- HTML listing of package listing on box

2.4.1 List package updates

Use Case Name:	**List package updates**
Description:	Generate a list of updates available for installed packages.
Actors:	User
Preconditions:	Package install list and list of available updates from network are available.
Triggers:	abt show-updates
Basic Course of Events:	1. User submits an updates request. 2. Package install list is processed to check each package for updates. 3. Available updates retrieved from over network. 4. User is notified by showing available updates for all installed packages.
Exceptions:	Warning when available update is older than installed package version.
Postconditions:	Package updates list has been generated.

2.4.2 HTML package listing

Use Case Name:	**HTML package listing**
Description:	Generate HTML formatted listing of installed packages.
Actors:	User
Preconditions:	Package install list is available.
Triggers:	abt html
Basic Course of Events:	1. User submits a html request. 2. Package install list is processed. 3. HTML page is generated showing installed packages list. 4. User is notified of HTML page location.
Exceptions:	none.
Postconditions:	HTML page showing package list has been generated.

2.5 Downloads

This section details the items that involve downloading diverse components for *AbTLinux* machines:

- pull package sources
- pull package tree
- pull patches
- pull news feed

2.5.1 Pull package sources

Use Case Name:	**Pull package sources**
Description:	Downloading the source tarball for a given package.
Actors:	User
Preconditions:	Package description and network are available.
Triggers:	abt download <pkg>
Basic Course of Events:	1. User requests for package source(s) to be downloaded. 2. The source location is obtained from the package description. 3. Support for http(s), (s)ftp, cvs, subversion and tla repositories. 4. Source is downloaded and placed in download location.
Exceptions:	Source location is unreachable or gives error, report error.
Postconditions:	Package source(s) have been downloaded.

2.5.2 Pull package tree

Use Case Name:	**Pull package tree**
Description:	Downloading a package tree, optionally by tree name.
Actors:	User
Preconditions:	Network is available.
Triggers:	abt update [name]
Basic Course of Events:	1. User requests for package tree to be downloaded. 2. Optional name given, then look up the location in registered list (internal). 3. Support for http(s), (s)ftp, cvs, subversion and tla repositories. 4. Package tree is downloaded and installed.
Exceptions:	Source location is unreachable or gives error, report error.
Postconditions:	Package tree has been downloaded and installed.

2.5.3 Pull patches

Use Case Name:	**Pull patches**
Description:	Downloading available patches, optionally by repository name.
Actors:	User
Preconditions:	Network is available.
Triggers:	abt patches [name]
Basic Course of Events:	1. User requests for patches to be downloaded. 2. Optional name given, then look up the location in registered list (internal). 3. Support for http(s), (s)ftp, cvs, subversion and tla repositories. 4. Patches are downloaded and applied.
Exceptions:	Source location is unreachable or gives error, report error.
Postconditions:	Patches have been downloaded and applied.

2.5.4 Pull news feed

Use Case Name:	**Pull news feed**
Description:	Downloading and view available news feed.
Actors:	User
Preconditions:	Network is available.
Triggers:	abt news
Basic Course of Events:	1. User requests abt news feed download. 2. Retrieve abt news feed file. 3. Save and display news feed.
Exceptions:	News feed location is unreachable or gives error, warning and attempt to display old news feed file.
Postconditions:	News feed has been downloaded and displayed.

2.6 Logging

This section details the requirements for keeping track of work progress with some form of logging. These items are the ones that require tracking:

- log package installation, not tracking:
  - /home
  - /tmp
  - /var/tmp
- log package files integrity information (tool to use: md5)
- log package builds
- log general progression (journal)
- cached builds include all scripts needed to build so it can always be duplicated (not lost on package maintenance in the future), track the following for each package:
  - Major version number
  - Build time (time installed package was built)
  - Configuration time (time configuration last updated)
  - Dependencies

2.6.1 Log package install

Use Case Name:	**Log package install**
Description:	What will be logged from a package install.
Actors:	System
Preconditions:	none.
Triggers:	The install phase of a package install.
Basic Course of Events:	1. A package will reach the *Preinstall* section of package install. 2. *Install* section of package is processed. 3. A list of installed files is logged. 4. File in following directories are not tracked: (a) home directories. (b) tmp directories. (c) var/tmp directories.
Exceptions:	Any failure to create log files will be reported as error.
Postconditions:	Package files installed have been tracked in log file.

2.6.2 Log package files integrity

Use Case Name:	**Log package files integrity**
Description:	A log will be created with each file a package installs being integrity checked.
Actors:	System
Preconditions:	Integrity software is available on system.
Triggers:	The end of install phase of a package install.
Basic Course of Events:	1. After files installed, the installed files log is parsed to create a checksum for each file. 2. A file can be marked as a configuration file and not tracked. 3. A file can be marked as shared and this will be noted in log. 4. A log is created containing the resulting integrity information.
Exceptions:	Any failure to create log files will be reported as error.
Postconditions:	Package files integrity information has been logged in a file.

2.6.3 Log package build

Use Case Name:	**Log package build**
Description:	What will be logged from a package build.
Actors:	System
Preconditions:	none.
Triggers:	The build phase of a package install.
Basic Course of Events:	1. *Build* section of package is processed. 2. The running of the package build is logged to a file.
Exceptions:	Any failure to create log files will be reported as error.
Postconditions:	Package build has been logged in a file.

2.6.4 Log journal

Use Case Name:	**Log journal**
Description:	Provide a journal of general abt activity.
Actors:	System
Preconditions:	none.
Triggers:	Any action abt does that is needing to be logged.
Basic Course of Events:	1. Any item needing to be logged will be appended to the journal with timestamp. 2. Auto journal rotation will be done at user defined number of lines or file size.
Exceptions:	If activity journal does not exist, it will be created.
Postconditions:	Abt is able to log activity to the journal.

2.6.5 Log package cache

Use Case Name:	**Log package cache**
Description:	A complete cache of a built package, including everything needed to duplicate the build at a later date.
Actors:	System
Preconditions:	none.
Triggers:	The end of a package install.
Basic Course of Events:	1. At end of a complete package install a cache will be made, including: 2. Copy of build configuration (preserve configuration time). 3. Copy of build log (preserve build time). 4. Copy of install log. 5. Copy of package description (includes major version number). 6. Copy of source. 7. Copy of dependency tree.
Exceptions:	Any failure to create cache will be reported as error.
Postconditions:	Package build has been cached.

2.7 Fixing

This section details requirements related to checking the health of *AbTLinux* machines and repairing any problems found:

- remove unused package sources
- remove unused package logs
- verify installed package files
- verify installed package symlinks
- verify installed package dependencies (libraries, versions)
- verify system integrity
- fix package(s)

2.7.1 Purge old package sources

Use Case Name:	**Purge old package sources**
Description:	Source tarballs will be removed for packages that are no longer installed. This will remove source tarballs for any versions of installed packages other than those currently installed.
Actors:	User
Preconditions:	Installed package list is available.
Triggers:	abt purge-src
Basic Course of Events:	1. The user requests to remove unused package sources from system. 2. The installed package list is used to determine which package (versions) to keep. 3. All source tarballs from packages not currently installed are removed.
Exceptions:	none.
Postconditions:	All old source tarballs have been removed from the system.

2.7.2 Purge old logs

Use Case Name:	**Purge old logs**
Description:	This command will query the user about removal of all old package logs (configuration, build, install, integrity) and cache logs for packages that are no longer installed.
Actors:	User
Preconditions:	Installed package list is available.
Triggers:	abt purge-logs
Basic Course of Events:	1. The user requests to remove old logs from the system. 2. The user is queried whether cache logs also need to be removed. 3. The installed package list is used to determine which package (versions) to keep. 4. All package logs are removed from the system for packages not currently installed. 5. Eventual cache logs are removed from the system for packages not currently installed.
Exceptions:	none.
Postconditions:	All old package log and eventual cache logs have been removed from the system.

2.7.3 Verify installed package files

Use Case Name:	**Verify installed package files**
Description:	Verify that the given packages files are available on the system.
Actors:	User
Preconditions:	Package to be verified has a valid install log.
Triggers:	abt verify-files <pkg>
Basic Course of Events:	1. User submits a verify files request. 2. Installed files for given package are checked if they exist by comparison to packages install log. 3. After package checked, overview of eventual missing files reported to user.
Exceptions:	none.
Postconditions:	The installed files for given package have been verified, eventual missing files have been reported.

2.7.4 Verify installed package symlinks

Use Case Name:	**Verify installed package symlinks**
Description:	Verify that the given packages symlinks are available on the system.
Actors:	User
Preconditions:	Package to be verified has a valid install log.
Triggers:	abt verify-symlinks <pkg>
Basic Course of Events:	1. User submits a verify symlinks request. 2. Installed symlinks for given package are checked if they exist by comparison to packages install log. 3. After package checked, overview of eventual missing symlinks reported to user.
Exceptions:	none.
Postconditions:	The installed symlinks for given package have been verified, eventual missing symlinks have been reported.

2.7.5 Verify installed package dependencies

Use Case Name:	**Verify installed package deps**
Description:	Verify that the given packages dependencies are not broken (libraries).
Actors:	User
Preconditions:	Package to be verified has a valid description.
Triggers:	abt verify-libs <pkg>
Basic Course of Events:	1. User submits a verify libraries request. 2. Dependency engine used to determine given packages dependency tree. 3. Installed files for each dependency are checked for existence. 4. Installed symlinks for each dependency are checked for existence. 5. After package checked, overview of eventual broken libraries reported to user.
Exceptions:	none.
Postconditions:	The installed libraries for given package have been verified, eventual discrepencies have been reported.

2.7.6 Verify installed package integrity

Use Case Name:	**Verify installed package integrity**
Description:	Verify given packages files integrity.
Actors:	User
Preconditions:	Package to be verified has a valid install log and md5 is available.
Triggers:	abt verify-integrity <pkg>
Basic Course of Events:	1. User submits a verify integrity request. 2. Integrity of installed files for given package are checked by comparison to packages install log. 3. After package checked, overview of eventual tainted files reported to user.
Exceptions:	none.
Postconditions:	The integrity of installed files for given package have been checked, eventual tainted files have been reported.

2.7.7 Fix package

Use Case Name:	**Fix package**
Description:	Check the package for missing/broken libraries, symlinks and file permissions.
Actors:	User
Preconditions:	Package to be fixed is installed.
Triggers:	abt fix [pkg]
Basic Course of Events:	1. User submits a fix request, either on a package or on entire installed packages list if no package is given. 2. Package(s) are checked by comparing: (a) Installed files are checked if they exist. (b) Installed files are checked for integrity. (c) Installed libraries are checked for dependency breakage. (d) Configuration files are checked only for existence. 3. Any problems found result in User being asked if rebuild wanted. 4. Default if User does not respond is to rebuild package. 5. Check for possible dependency breakage. 6. After package(s) checked, overview of good, bad and fixed package(s) reported to User.
Exceptions:	Any problems results in package(s) being left in the install or uninstall queues.
Postconditions:	1. User has report of good, bad and fixed package(s).

2.8 Patching

This section details the requirements related to providing patching for updates and fixes on *AbTLinux* machines:

- provide for updating with a patch per package
- provide package tree patches (i.e. for entire stable tree)

2.8.1 Patch package update

Use Case Name:	**Patch package update**
Description:	Patches will be provided as preferred method of updating a package, fall-back is to use the complete updated package description.
Actors:	User
Preconditions:	Patch is available over the network and package is installed.
Triggers:	Any command leading to update or fix from AbTLinux network.
Basic Course of Events:	1. User request leading to an update or fix being needed over the network. 2. Attempt is made to retrieve needed patch over network. 3. Fall-back is to retrieve needed package (entire package description) over network. 4. Package is updated with patch or new package description and left in install queue.
Exceptions:	Patch fails to be applied, fall-back to entire package description update.
Postconditions:	Package has been updated with patch if available, otherwise with complete package listing.

2.8.2 Patch package tree update

Use Case Name:	**Patch package tree update**
Description:	Patches will be provided as preferred method of updating a package tree, fall-back is to use the complete updated package tree.
Actors:	User
Preconditions:	Patch is available over the network.
Triggers:	Any command leading to update of the entire package tree from AbTLinux network.
Basic Course of Events:	1. User request leading to an update of the entire package tree over the network. 2. Attempt is made to retrieve the tree patch over network. 3. Fall-back is to retrieve entire tree over network. 4. Package tree is updated with patch or entire new tree.
Exceptions:	Patch fails to be applied, fall back to entire new tree.
Postconditions:	Package tree has been updated with patch if available, otherwise with complete new tree.

2.9 Maintaining

This section details the requirements related to maintenance tasks on *AbTLinux* machines:

- provide for central build box (clients pull package builds from central box)
- provide for specification of package sources download location

2.9.1 Central build

Use Case Name:	**Central build**
Description:	Allows a single machine to function as a build center, providing cached software package builds for other machines on a network.
Actors:	User
Preconditions:	Network is available.
Triggers:	User setup is pointing to this central server as provider of cached software packages and requests a package install or update.
Basic Course of Events:	1. User request leading to an (re)-install, update or fix of package being needed. 2. User setup to direct all request to remote machine for cached packages. 3. Fall-back is to search for cached packages on local machine. 4. Cached package is retrieved and installed on local machine. 5. A fix package check is run. 6. User notified that package installed successfully.
Exceptions:	Unable to obtain cached package from central server, fall-back to local machine.
Postconditions:	Cached package has been installed and checked successfully.

2.9.2 Source location package tree

Use Case Name:	**Source location package tree**
Description:	User can specify package tree repository, default being the AbTLinux provided repository.
Actors:	User
Preconditions:	Given repository is available and reachable.
Triggers:	User requests to set a package tree repository location.
Basic Course of Events:	1. User provides a given URI as default package tree repository. 2. Fall-back is to use the AbTLinux provided repository. 3. User requests package tree update. 4. Package tree is updated from given URI. 5. User notified that package tree updated successfully.
Exceptions:	Unable to obtain package tree from URI, fall-back to AbTLinux repository.
Postconditions:	Package tree has been updated successfully.

2.10 Dependencies

This section will be covered in an apart requirements document, focusing on the entire dependency engine problem. This will be a student project for the course Requirements Engineering at the Radboud University Nijmegen.

3 Scenarios

Here I will work out the use scenarios into example scenarios filled with actual data.

3.1 Packages

Use Case Name:	**Install package**
Use Case Steps:	1. Installing a standard package for first time.
Alternative Path:	1. Installing package that has been previously built (cached).

Use Case Name:	**Reinstall package**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Remove package**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Downgrade package**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Freeze package**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.2 Queries

Use Case Name:	**Show package details**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show installed packages**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show package install**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show package build**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show frozen packages**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show package dependencies**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show untracked files**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show journal**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show file owner**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Search package descriptions**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show install queue**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Show available patches**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.3 Generation

Use Case Name:	**List package updates**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**HTML package listing**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.4 Downloads

Use Case Name:	**Pull package sources**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Pull package tree**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Pull patches**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Pull new feed**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.5 Logging

Use Case Name:	**Log package install**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Log package file integrity**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Log package build**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Log journal**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Log package cache**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.6 Fixing

Use Case Name:	**Purge old package sources**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Purge old logs**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Verify installed package files**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Verify installed package symlinks**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Verify installed package deps**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Verify installed package integrity**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	Fix package
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.7 Patching

Use Case Name:	**Patch package update**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	Patch package tree update
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

3.8 Maintaining

Use Case Name:	**Central build**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

Use Case Name:	**Source location package tree**
Use Case Steps:	1. Steps
Alternative Path:	1. Steps

4 Design

Here I will detail the design and implementation choices made for *AbTLinux*. Some early thoughts are Object Orientation and Ruby as implementation language.

4.1 Package structure

A single package will have one file containing the entire structure needed for installing the software it offers. This structure will be split into sections, such as the following:

- details{}
- pre{}
- configure{}
- pre-build{}
- build{}
- pre-install{}
- install{}
- post{}

This is rather flexible and open for debate, an example should be put together for a rather complex package just to give an idea of what it will look like. I think each section should be enclosed in curly brackets.

4.1.1 Install locations

Base system for AbTLinux will be in /usr, all others will be installed into /usr/local/*. This will facilitate ease of usage within other Linux systems that follow the LSB/FHS.

4.2 Configuration update tools

Here I mean tools dealing with how we want configuration file updates to be handled. Almost everything located in /etc of the Linux filesystem is considered holy to a running system. They need to be updated sometimes when a new package version is installed, but should never destroy an existing configuration. We want to provide for the following:

- view existing configuration file
- view new (default to install) configuration file
- allow editing to take place in old or new config file
- show user diff
- let user select one of the above options to install

4.3 Package manager

5 Dictionary

A list can be found here that contains terms and their AbTLinux related definitions. Hopefully this will keep everyone on the same page.

- **Package** - this will contain all needed information for the abt tool to install a single piece of software.
- **abt** - the package manager tool that can be run from the command line and provides for all software management on the AbTLinux distribution.
- **depEngine** - the dependecny engine, part of the tool set used by 'abt' to facilicate package management.
- **Dependency** - a package X will have a depenceny on another package Y if X needs a service provided by Y. X is then dependent on Y.
- **AbTLinux** - the name of this distribution, derived from ABout Time Linux.
- **Journal** - the running log of abt actions, where 'abt' will provide information as to actions taken on the AbTLinux system.

6 Thanks

This is to thank those whose input and critical eyes have been on this document since the early days of its conception. I will list them in alphabetical order:

- Bas van Gils for his proof reading and insistent nature that keeps me busy.
- Benoit Papillault for his freely given time and support with all projects we have worked on together.
- Jose Silva, another critical eye and stable spirit in the often turbulent world of free software development.
- Tony Smith who has been my verbal Linux sparring partner for a long time.
- Laurent Wandrebeck who is a pretty sharp DB cookie and long time partner in Linux crimes.

Thanks guys!

Special mention: the Source Mage GNU/Linux team - For good or bad my experiences with you taught me much, for that I am forever greatfull. You are always welcome in my world!

References

Easinstaller (2004). *Easinstaller website*.
`http://easinstaller.sourceforge.net`. last checked 23 Sep, 2004.

micropkg (2004). *The Micro Package Manager website*.
`http://u-os.org/upm.html`. last checked 23 Sep, 2004.

SMGL (2004). *Source Mage GNU/Linux website*.
`http://www.sourcemage.org`. last checked 23 Sep, 2004.