This page is under construction.
You are welcome to assist in its construction by editing it as well.
This message should be deleted after editing.

Contents

- 1 Basics
  - 1.1 Packager's point of view
  - 1.2 Packaging Mechanism
  - 1.3 Pardus SVN Repository
  - 1.4 File and Directory Locations
  - 1.5 Structure of a PiSi Package
- 2 PiSi Package Building
  - 2.1 pspec.xml
    - 2.1.1 Source part
    - 2.1.2 Package part
    - 2.1.3 History part
  - 2.2 actions.py
    - 2.2.1 Actions API
- 3 Directories (comar and files)
  - 3.1 comar directory
    - 3.1.1 package.py
    - 3.1.2 service.py
    - 3.1.3 Adding comar scripts to package
  - 3.2 files directory
- 4 Other files
  - 4.1 translations.xml
  - 4.2 components.xml
- 5 Example

PİSİ (Packages Installed Successfully as Intended) is a binary package management system which have been developed within Pardus project. PİSİ has more function than just installing/uninstalling packages. For example, it is possible to regain old package setup automatically. Packages need to be processed as pisi packages and presented as binary packages for using them as pisi packages. For detailed information you can look at Pardus official site

Basics

Package: A program, document or any kind of data for installing to system

Source file: A file that includes files which belongs to package and generally presented as archive files and generally downloadable via internet. Programs generally uses files that are archived as .tar.gz files. Archived files need to include program's non-compiled source code, data that program needs and documents.

Compiling: Operation that needs to be done to make executable binary files of programs that is written in compilable programing language like C,C++.

Dependency: Other packages that program uses while it is running or compiling.

Integrity code: This code is being used to verify that any downloaded data has been successfully downloaded without getting damaged. Sites generally put integrity codes obtained by md5 or sha algorithms for source files. Verifying is done by comparing this code and the code obtained from downloaded data.

Here can you download a English pisi gude:

http://www.pardususer.de/index.php?action=repository&dir=/pisi-howto

Packager's point of view

Adding package to pisi system is more than just making a pisi package. It is a process including gathering information about the package you want to make pisi package of, testing and debugging your pisi package, keeping up with newer versions. A user should start building pisi packages only if user thinks s/he can follow this process. Pisi packages are like the children of the packager and hets replenished with every single debugging and version change. However, unsupervised packages can't live long in the linux world which is growing rapidly. People shouldn't make pisi packages as many as they can, but should make pisi packages as many as they can keep track of. And also don't forget to share your packages from the internet as Linux world is one of the best examples of collective working. By doing so, you can also find other users who can help you with debugging and keeping track of your package.

Packaging Mechanism

This mechanism is a mechanism that makes binary packages that pisi uses by using source codes. If we omit technical details, process is like following:

- Source codes get downloaded by processing "http://en.pardus-wiki.orgSource"http://en.pardus-wiki.org part of the pspec.xml file and after it gets verified, it gets extracted to /var/pisi/package_name/work
- To obtain binary data action.py gets processes and installation is made to /var/pisi/package_name/install using source code.
- If there is any additional file they get coppied to /install directory as it defined at pspec.xml
- Pisi makes index file for the files in the /install directory. This index is compatible with pisi database and in XML format.
- Index and binary get combined with datas obtained from pspec.xml and archived as .pisi package

As one can see, pisi installs package to /var/pisi/paket_adi/install while it produces .pisi file. Pisi behaves this

directory as it were root system. By examining this directory one can monitor which files go to which directory.

Pardus SVN Repository

Pardus SVN repository is a kind of library that you can examine pardus projects and the source files of pisi packages that developers made. You can check how developers made pisi packages for similar packages that you want to make a pisi package for. You can access this repository from here. At that site, there is pisi source files of pisi packages. All answers to "http://en.pardus-wiki.orghow"http://en.pardus-wiki.org questions that new beginners ask actually can be found in source files available at Pardus SVN repository.

File and Directory Locations

Mandatory packaging files that pisi requires are pspec.xml and actions.py. Both should be in same directory. "http://en.pardus-wiki.orgcomar"http://en.pardus-wiki.org and "http://en.pardus-wiki.orgfiles"http://en.pardus-wiki.org directories can be used if necessary. These directories must be in same directory as pspec.xml. Source files looks like this:

```
package_name
      |__pspec.xml
      |__actions.py
      |__files
      |__comar
            |__package.py
            |__service.py
```

Structure of a PiSi Package

A pisi package is essentially a zipped file. Let's download one and examine it. To do so write following to your konsole:

```
$ wget http://paketler.pardus.org.tr/pardus-2008/knazar-0.2_p21206-7-3.pisi
```

Now that our pisi package downloaded to our current working directory, we can extract files from our pisi package. Open pisi package with "http://en.pardus-wiki.orgark"http://en.pardus-wiki.org and select zip archive if it asks for what kind of archive it is. After that extraxt it somewhere. You'll see there are 3 files exist: files.xml, metadata.xml and install.tar.lzma.

files.xml contains locations, types, sizes and sha1sums of all files in the package and looks like this:

```
<Files>
    <File>
        <Path>path/to/install/the_file.txt</Path>
        <Type>data</Type>
        <Size>1602</Size>
        <Uid>0</Uid>
        <Gid>0</Gid>
```

```
        <Mode>0644</Mode>
        <Hash>a30a14bca4a66f2519e2857027b00c31d72e3896</Hash>
    </File>
                            .
                            .
                            .
    <File>
        <Path>path/to/install/the_file.bin</Path>
        <Type>data</Type>
        <Size>1602</Size>
        <Uid>0</Uid>
        <Gid>0</Gid>
        <Mode>0644</Mode>
        <Hash>a30a14bca4a66f2519e2857027b00c31d72e3896</Hash>
    </File>
</Files>
```

metadata.xml contains general information like package name, homepage, packager, etc. Looks just like pspec.xml file.

install.tar.lzma contains compressed files that we can extract and have a look at. Do not hesitate to check them so you can get better understanding of a pisi package.

Here can you download a English pisi guide to read and print out:

http://www.pardususer.de/index.php?action=repository&dir=/pisi-howto

PiSi Package Building

In order to build a pisi package we need to prepare at least two files by hand: *pspec.xml* and *actions.py*

pspec.xml

This file is about definition and content of package and packaging progress. XML files are like a kind of database. An XML file which has a data and a parameter which belongs to that data looks like this:

```
<Path fileType="http://en.pardus-wiki.orgexecutable"http://en.pardus-wiki.org>/usr/kde/3.5/bin</Pa
    |      |       |                  |
    |      |       |                  \--Data
    |      |        \--------------------Property of data
    |       \----------------------------Name of property
     \-----------------------------------Tag
```

Basic pspec.xml structure is like following and first two lines are standard meaning you have to keep them as they are.

```
<?xml version="http://en.pardus-wiki.org1.0"http://en.pardus-wiki.org ?>
<!DOCTYPE PISI SYSTEM "http://en.pardus-wiki.orghttp://www.pardus.org.tr/projeler/pisi/pisi-spec.d
<PISI>
```

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

4cs, 6502acme, 6502kickass, 6502tasm, 68000devpac, abap, actionscript, actionscript3, ada, algol68, apache, applescript, apt_sources, asm, asp, autoconf, autohotkey, autoit, avisynth, awk, bascomavr, bash, basic4gl, bf, bibtex, blitzbasic, bnf, boo, c, c_loadrunner, c_mac, caddcl, cadlisp, cfdg, cfm, chaiscript, cil, clojure, cmake, cobol, coffeescript, cpp, cpp-qt, csharp, css, cuesheet, d, dcs, delphi, diff, div, dos, dot, e, ecmascript, eiffel, email, epc, erlang, euphoria, f1, falcon, fo, fortran, freebasic, fsharp, gambas, gdb, genero, genie, gettext, glsl, gml, gnuplot, go, groovy, gwbasic, haskell, hicest, hq9plus, html4strict, html5, icon, idl, ini, inno, intercal, io, j, java, java5, javascript, jquery, kixtart, klonec, klonecpp, latex, lb, lisp, llvm, locobasic, logtalk, lolcode, lotusformulas, lotusscript, lscript, lsl2, lua, m68k, magiksf, make, mapbasic, matlab, mirc, mmix, modula2, modula3, mpasm, mxml, mysql, newlisp, nsis, oberon2, objc, objeck, ocaml, ocaml-brief, oobas, oracle11, oracle8, oxygene, oz, pascal, pcre, per, perl, perl6, pf, php, php-brief, pic16, pike, pixelbender, pli, plsql, postgresql, povray, powerbuilder, powershell, proftpd, progress, prolog, properties, providex, purebasic, pycon, python, q, qbasic, rails, rebol, reg, robots, rpmspec, rsplus, ruby, sas, scala, scheme, scilab, sdlbasic, smalltalk, smarty, sql, systemverilog, tcl, teraterm, text, thinbasic, tsql, typoscript, unicon, uscript, vala, vb, vbnet, verilog, vhdl, vim, visualfoxpro, visualprolog, whitespace, whois, winbatch, xbasic, xml, xorg_conf, xpp, yaml, z80, zxbasic

```
  <Package>
       ...
  </Package>

   <History>
       ...
   </History>

</PISI>
```

Lots of tags exist at these parts of pspec.xml and some of them are special. Possible datas that these special tags can hold are predefined so that they can be identified by Pardus system. Values that these tags can hold and information about usage of these tags are gathered in a Relax NG (rng) file by Pardus developers. These rng files are scheme files that can be used for getting information about tags within a XML file. Every tag is defined here within a scheme. rng files has a structure that can easily get understood because this structure is same as a XML structure.

Most up-to-date version of rng scheme of pspec.xml is accessible via http://svn.pardus.org.tr/uludag/trunk/pisi/pisi-spec.rng. Easiest way to find information about a tag is to make a search for <!-- tag -->

Source part

This part gives basic information about package to pisi. In addition, which category and part that package belongs in Pardus system structure, which packages need to exist while compiling, which patches need to applied before compiling process are also defined here. Basically this part looks like this:

Invalid language.

You need to specify a language like this: `<source lang="html4strict">...</source>`

Supported languages for syntax highlighting:

`4cs`, `6502acme`, `6502kickass`, `6502tasm`, `68000devpac`, `abap`, `actionscript`, `actionscript3`, `ada`, `algol68`, `apache`, `applescript`, `apt_sources`, `asm`, `asp`, `autoconf`, `autohotkey`, `autoit`, `avisynth`, `awk`, `bascomavr`, `bash`, `basic4gl`, `bf`, `bibtex`, `blitzbasic`, `bnf`, `boo`, `c`, `c_loadrunner`, `c_mac`, `caddcl`, `cadlisp`, `cfdg`, `cfm`, `chaiscript`, `cil`, `clojure`, `cmake`, `cobol`, `coffeescript`, `cpp`, `cpp-qt`, `csharp`, `css`, `cuesheet`, `d`, `dcs`, `delphi`, `diff`, `div`, `dos`, `dot`, `e`, `ecmascript`, `eiffel`, `email`, `epc`, `erlang`, `euphoria`, `f1`, `falcon`, `fo`, `fortran`, `freebasic`, `fsharp`, `gambas`, `gdb`, `genero`, `genie`, `gettext`, `glsl`, `gml`, `gnuplot`, `go`, `groovy`, `gwbasic`, `haskell`, `hicest`, `hq9plus`, `html4strict`, `html5`, `icon`, `idl`, `ini`, `inno`, `intercal`, `io`, `j`, `java`, `java5`, `javascript`, `jquery`, `kixtart`, `klonec`, `klonecpp`, `latex`, `lb`, `lisp`, `llvm`, `locobasic`, `logtalk`, `lolcode`, `lotusformulas`, `lotusscript`, `lscript`, `lsl2`, `lua`, `m68k`, `magiksf`, `make`, `mapbasic`, `matlab`, `mirc`, `mmix`, `modula2`, `modula3`, `mpasm`, `mxml`, `mysql`, `newlisp`, `nsis`, `oberon2`, `objc`, `objeck`, `ocaml`, `ocaml-brief`, `oobas`, `oracle11`, `oracle8`, `oxygene`, `oz`, `pascal`, `pcre`, `per`, `perl`, `perl6`, `pf`, `php`, `php-brief`, `pic16`, `pike`, `pixelbender`, `pli`, `plsql`, `postgresql`, `povray`, `powerbuilder`, `powershell`, `proftpd`, `progress`, `prolog`, `properties`, `providex`, `purebasic`, `pycon`, `python`, `q`, `qbasic`, `rails`, `rebol`, `reg`, `robots`, `rpmspec`, `rsplus`, `ruby`, `sas`, `scala`, `scheme`, `scilab`, `sdlbasic`, `smalltalk`, `smarty`, `sql`, `systemverilog`, `tcl`, `teraterm`, `text`, `thinbasic`, `tsql`, `typoscript`, `unicon`, `uscript`, `vala`, `vb`, `vbnet`, `verilog`, `vhdl`, `vim`, `visualfoxpro`, `visualprolog`, `whitespace`, `whois`, `winbatch`, `xbasic`, `xml`, `xorg_conf`, `xpp`, `yaml`, `z80`, `zxbasic`

IsA tag and PartOf tag are the tags that people mostly waver about.

IsA tag required to determine which category that package belongs so that packages belong to same category can be listed together using a interface like package-manager. This tag can be defined more than once. You can check for available datas that this tag can hold in pisi-spec.rng file.

PartOf tag is for determining which component that package belongs at the system. So that users can make mass operations about a component group. For example, you can install all of kde4 related packages using:

```
$ sudo pisi it -c desktop.kde4
```

To see all component group names and descriptions write this to your console:

```
 $ pisi list-components
```

You can decide which group does your package belong checking the output of code above.

If integrity code of source code cannot be found on web-site of package, you can find it using "http://en.pardus-wiki.orgsha1sum file_name"http://en.pardus-wiki.org code.

Package part

This part holds definitions about pisi files that will created after packaging. If it is required to create more than one pisi file from a package, every one of them should be written in the same pspec.xml file but into different Package part for each. Creating pisi files is done according to this parts.

```
<Package>
   <Name>name of pisi oackage</Name>
   <RuntimeDependencies>
```

```
        <Dependency versionFrom="http://en.pardus-wiki.orgversion number"http://en.pardus-wiki.org>
        <Dependency></Dependency>
    </RuntimeDependencies>
    <Files>
        <Path fileType="http://en.pardus-wiki.orgtype"http://en.pardus-wiki.org>directory to instal
        <Path></Path>
    </Files>
</Package>
```

Most important part here is "http://en.pardus-wiki.orgFiles"http://en.pardus-wiki.org part. As mentioned before, source file gets compiled after patched if necessary and become installed to /var/pisi/package_name/install. Directories that Path tag uses should be this directories. Contents of directories mentioned here will be added to pisi file by pisi. Directories that didn't mentioned here won't be added to pisi file. However, while this files become added to directories, they gets added according to their "http://en.pardus-wiki.orgfileType"http://en.pardus-wiki.org property. You can find all available fileType properties here

History part

```
<History>
    <Update release="http://en.pardus-wiki.orgRelease number"http://en.pardus-wiki.org>
        <Date>Release date</Date>
        <Version>Version date</Version>
        <Comment>Brief explanation about changes made</Comment>
        <Name>name of updater</Name>
        <Email>e-mail address of updater</Email>
    </Update>
    <Update>
         ...
    </Update>
</History>
```

Information about updates and updaters are registered here by update parts every updater added here.

Important part here is release number. Sometimes developers of packages won't increase version number when they fix little bugs or patched the packages. And sometimes you need to patch source code to increase compatibility to your system. This means version number of the package will stay the same but you need to make a new release. This release number will be added after the package name and prevent the confusion. Release number increase by one with every different release.

actions.py

Actions API

Actions API has variety of functions that it cannot be described here one by one. Developers made documentation for it: Actions_API

Directories (comar and files)

There are two directories described as comar and files to make available the files that package or system needs. They are not mandatory. You can add them if package needs them.

comar directory

This directory keeps files that is produced to registered to COMAR system which is one of the basic components of Pardus. This directory can hold package.py and service.py to meet the needs of package.

package.py

COMAR operates some functions from this script when it is needed to operate specific operations after installation of before uninstallation. If operations need to be done after installation you should use postInstall function and if operations need to be done before uninstallations you can should use preRemove function. This file basically looks like this:

```
#!/usr/bin/python

def postInstall():
   ...

def preRemove():
    ...
```

service.py

it is a script that COMAR uses for applications which will operate as service. When it is needed to gather information about starting, stopping or situation of the service, related functions from this file get operated. It looks like following:

```
#!/usr/bin/python

from comar.service import *

serviceType = "http://en.pardus-wiki.orgserver"http://en.pardus-wiki.org
serviceDesc = _({"http://en.pardus-wiki.orgen"http://en.pardus-wiki.org: "http://en.pardus-wiki.or
                 "http://en.pardus-wiki.org(code of another language)"http://en.pardus-wiki.org: "

@synchronized
def start():
    ...

@synchronized
def stop():
    ...

def status():
    ...
```

Adding comar scripts to package

Pisi system does not automatically adds scripts in COMAR directory to package. You should make an addition to <Package> part of your pspec.xml like following:

```
<Provides>
   <COMAR script="http://en.pardus-wiki.orgpackage.py"http://en.pardus-wiki.org>System.Package</CO
   <COMAR script="http://en.pardus-wiki.orgservice.py"http://en.pardus-wiki.org>System.Service</CO
</Provides>
```

files directory

If there is a patch for source file, it must be placed at this directory. Also the files which source package doesnt contain, but application needs for running, are located here. Like the application's icon at programs menu, etc... Besides this there can be files despite located at source pack, we have changed to run properly at our system. For example the config file of application or etc... Ayrıca kaynak pakette olmayan ama programın çalışması için gerekli olan dosyalar da burada bulunur. Örneğin programın Programlar menüsündeki ikonu gibi. Bunun yanında kaynak pakette olan ama programın bizim sistemimize uygun çalışması için değiştirdiğimiz dosyalar da olabilir. Mesela programın kaynak paketinden çıkan ayar dosyası o haliyle programdaki pek çok özelliği devre dışı bırakmış olabilir veya bizim sistemimize uygun değildir ancak içerdiği pek çok açıklama da kullanıcılar için gerekli olabilir. Bu durumda o dosyanın yanında bizim üzerinde değişiklik yaptığımız dosyanın da pakete eklenmesi için değiştirdiğimiz dosyayı bu dizine yerleştirmeliyiz.

IF there are patch files under this folder, for using all of those files by pisi, we must add parameters to Source section at pspec.xml file like under:

<Patches>

```
<Patch level="http://en.pardus-wiki.orgapp level"http://en.pardus-wiki.org>name of patch file</
<Patch>...</Patch>
```

</Patches>

For adding other files to this package, we must add all files to Packages section like under:

<AdditionalFiles>

```
<AdditionalFile owner="http://en.pardus-wiki.orguser"http://en.pardus-wiki.org permission="http
<AdditionalFile ... >...</AdditionalFile>
```

</AdditionalFiles>

For each file, we must use a new AdditionalFile tag at AdditionalFiles section. Target field must contain file path like "http://en.pardus-wiki.org/etc/samba/samba.conf"http://en.pardus-wiki.org. Owner and permission fields declerate user permission of this file. Owner is username of owner and permission is chmod code.

Other files

translations.xml

This file is an internationalization file for pspec.xml. We can write multi-lingual description and summary sections at this file. This file must be located at same folder with pspec.xml file. You an find an example trasnlation file here: http://svn.pardus.org.tr/pardus/2007/kernel/kernel/

components.xml

You can use this file instead of PartOf tag at pspec.xml file. This file must be located at parent directory of pspec.xml file. You can find an example file here http://svn.pardus.org.tr/pardus/2007/kernel/ . This file can be useful, if you have such packages. for example you can locate each package its own folder, so every folder can have different component.xml file.

Example