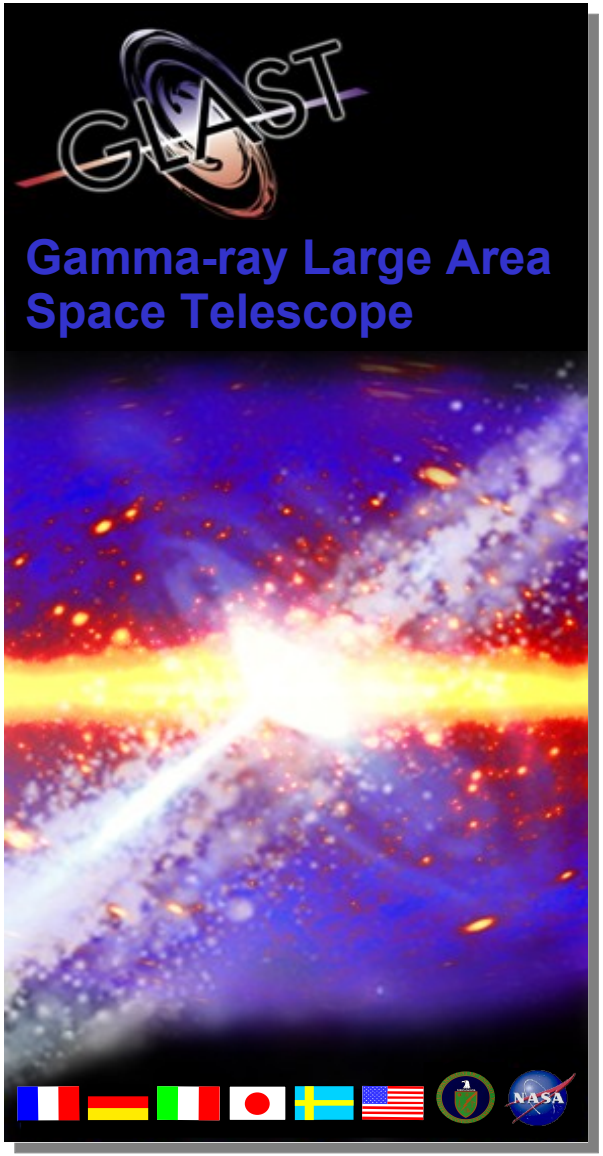


# Flight Ops Soft. package management

## Overview

D. Decotigny
March 23, 2007

# Outline

- FOS Wishes / constraints
- Some use cases
- Status
- Packages already available
- User perspective
- Packager perspective
- Installer perspective

# FOS wishes

**A system that should:**

- allow installation of individual packages anywhere
  - possibly multiple installations per user
  - without `root`-user privilege
- maintain a list of dependencies between:
  - local packages / system packages
  - local packages / local packages
- have a "moderate" (...) hard-disk space overhead
  - no `chroot` installation
- for developers/users: moderate/no "visible change"
  - automatic environment variable settings
- for packagers: moderate effort to learn the packaging tools, to package required 3rd party software

# Some use cases

- LICOS/ISOC on SLAC workstations (development):
  - AFS: a series of repositories of pre-compiled "3rd party" software + default stable ISOC/LICOS software
  - no constraint on the place to install them => "**relocatable**"
  - CVS local copies of LICOS/ISOC still possible (PYTHONPATH)
  - need to set up environment variables for this to work
- System-wide installation possible:
  - installation of 3rd party software in the default location, ("/usr"-style), or elsewhere if needed
  - no need to set up environment variables
- Workstations outside Slac (Glast collaboration):
  - users manage to install 3rd party software manually or by downloading the relocatable pre-compiled SW from Slac
  - they can also rebuild everything from the sources (long)

# Status

- **RPM** adopted, RHEL4 (SL4) base system installation assumed, limited support for RHEL3
  - 1 so-called “local ISOC environment” = the bin/, lib/.... + the RPM DB + RPM binary & sources packages
- The RPM DB reflects:
  - the set of system-wide installed packages
  - the set of locally installed packages
- a program (`slaverpmdb`) to synchronize the local RPMDB with the system-wide packages that might have been update/installed
- usual trick to setup environment to build RPMs (`%_topdir`) + custom rpm macros
- a wrapper script (`isoc`) that sets up the environment (`PATH, LD_LIB`...) to run rpm/rpmbuild and to launch programs in the local environment
- A series of modified **relocatable** standard packages (mostly from FC6) that can be built/installed in the local environment (libtool, db4, boost, python, qt, ...)
- 2 makefiles to automate the build and install phases of all these packages

# Packages already available

- Currently: only those needed by LICOS/ISOC
  - python 2.4.4 and 2.5
  - swig, Boost.Python (+ boost 1.33), sip
  - python numarray + numpy
  - FITS lib + 2 python bindings
  - qt 3.3.7 and 4.2.2, PyQt for both pythons
  - qwt3 and 4, PyQwt 5 for both pythons
  - doxygen + dot
  - mysql, pgsql, sqlite, db4, oracle support (incl. for python)
  - misc: Minuit2, sqlalchemy, db4, python xlrd, 4Suite XML, Cheetah, NetLogger
  - HippoDraw (+ sip & Boost.Python bindings)
  - FSW: CCSDS, DFI and VSC + python bindings
  - ISOC, LICOS
- Full list (with versions) + reference doc + usage notes:
  https://confluence.slac.stanford.edu/display/~decotigny/ISOC+RPM

# User perspective (0)
## *How does it look like ?*

- **Basic layout (single hierarchy):**

```
shell> cd /path/to/installed/environment
shell> ls
bin/  etc/  include/  lib/  sbin/  share/  src/  var/
shell> ls var/isoc_rpm
BUILD/  RPMS/  SOURCES/  SPECS/  SRPMS/  etc/  rpmdb/
```

- **"Scattered" layout (1 hierarchy / package), NON-standard:**

```
shell> ls
bin/  etc/  include/  lib/  sbin/  share/  src/  stow/  var/
shell> ls stow/
HippoDraw-1.14.8.7/              python24-sqlite2-2.3.2/
LICOS-docs-2.6.0/                python24-xlrd-0.5.2/
Minuit2-5.15.02/                 python25-2.5/
...
shell> ls stow/HippoDraw-1.14.8.7/
bin/  include/  lib/  share/
shell> ls -l bin/hippodraw
lrwxr-xr-x  1 decot gl 40 Mar 20 18:22 bin/hippodraw -> ../stow/HippoDraw-
1.14.8.7/bin/hippodraw
```

# User perspective (1)
## *Basics: command by command*

- I don't want to change my environment but I want to start my python script using the ISOC packages:
  - /path/to/isoc/installation/bin/isoc isoc_run python script.py
- Wait... this is python 2.5 ! I'd like to compare to ISOC's python 2.4:
  - .../bin/isoc isoc_run python2.4 script.py
- I want to start qt4 designer in the “ISOC” environment:
  - .../bin/isoc isoc_run –add-env=qt4 designer
- I made a mistake, my sources are for qt3:
  - .../bin/isoc isoc_run –add-env=qt3 designer
- I have to run this script which needs qt3 and qt4:
  - .../bin/isoc isoc_run –add-env=qt3 –add-env=qt4 python toto.py

# User perspective (2)

- How many "--add-env=..." will I need ?!
  - Currently: only needed for qt3, qt4 and ISOC/LICOS package specific environment variables
  - Everything else is directly available using the PATH, LD_LIB..., MANPATH, ... environment variables.

# User perspective (3)
## *Start a new terminal*

- I see, but all these “/path/to/bin/isoc” are painful. How can I simply start a terminal that is “ISOC”-ified ?
  - –/path/to/bin/isoc isoc_terminal
- Well, are you sure ? I still cannot access ISOC's python !
  - –You must add this line (3 lines for C shells) in your shell environment setup. They will change your environment only when you start an isoc terminal:

    [ -n "$ISOC_WRAPPER" ] && eval `$ISOC_WRAPPER isoc_env`
- Ok, now “python” is the ISOC python. But my qt4 script doesn't work ! Help !
  - –/path/to/bin/isoc isoc_terminal –add-env=qt4
- This is not satisfying. I already an ISOC terminal and I don't want to start a new one ! But I want my current ISOC terminal to integrate the qt4 config !
  - –. `pkg-config –variable=shenv qt4`

*Bourne shell examples only*

# User perspective (4)
## *Alter one's terminal environment*

*Bourne shell examples only*

- I am in Pisa, I want to have my terminal with the SLAC AFS ISOC environment... and starting an ISOC terminal is “not optimal”. Let me simply alter my current ssh session environment:
  - eval `/path/to/bin/isoc isoc_env`
- The same, with qt4 pre-integrated:
  - eval `/path/to/bin/isoc isoc_env –add-env=qt4`
- And what if I want to integrate qt3 afterwards ?
  - . `pkg-config –variable=shenv qt3`

# User perspective (5)
## *Special case of the '#!'*

- How should my python program start ?
  - If it is to be run inside an isoc environment:

    #! /usr/bin/env python

    Then call it with:

    - .../bin/isoc isoc_run ./script.py
    - or directly "./script.py" inside an ISOC terminal
- I want to run it from outside an ISOC env (eg GLAST pipeline). It starts with:

  #! /path/to/bin/isoc isoc_run --add-env=qt4 python

  But it doesn't run ! What should I write ?

  - Use "shisoc", which calls "/path/to/bin/isoc isoc_run" for you:

    #! /path/to/bin/shisoc --add-env=qt4 python

# Packager perspective

- Everything is based on RPM
  - large soft base available => large number of “spec” files already existing
  - **BUT:** not “relocatable” in general
    - assumes that destination dir == known at compilation time
  - A few changes need to be applied to the stock RPM “spec” files (eg. coming from FC6/FC7)
    - almost always the same set of changes
    - the most common changes: see end of reference documentation
- To generate an RPM:

```
/path/to/bin/isoc rpmbuild -ba package.spec
```

# Installer perspective

- Installing a package

  NEVER call rpm directly !

  /path/to/bin/isoc isoc_install [rpm_opts] package.rpm ...

- Uninstalling package

  NEVER call rpm directly !

  /path/to/bin/isoc isoc_uninstall [rpm_opts] package.rpm ...

- Query the RPM database

  NEVER call rpm directly !

  /path/to/bin/isoc rpm -qa ...

# Conclusion

- Possible to deploy packages anywhere (RHEL4 base “recommended”)
- One single, central “isoc” program (+shisoc), a small number of commands
  - one makefile to automate initial rpm installation
- Possible to have:
  - a single hierarchy installation
  - or have each package installed into a separate directory (+ stow to create the symlinks to bin, lib, ...) : almost OK
- At SLAC on AFS (/afs/slac/g/glast/isoc/flightOps/):
  - 3 installations for RHEL4 / gcc3.4
  - 1 installation for RHEL4 / gcc3.2 (limited RHEL3 support)
- 2 confluence page:
  - ref doc: https://confluence.slac.stanford.edu/display/~decotigny/ISOC+RPM
  - quick start guide: https://confluence.slac.stanford.edu/display/~decotigny/ISOC+RPM+Quick+Start

# Bonus slides

# Some 3rd party packaging solutions

- **python distutils:**
  - OK for python packages, weird for non-python packages (python, qt, boost, ...)
- **autopackage:**
  - \+ Seems to handle system-wide/per-user installations correctly, some projects use it
  - – Mid/long-term support ?
- **zero install:**
  - \+ Publishing system + update notifications
  - – Only per-user install, ignore most systemwide dependencies (include needed DLL)
- **klik:**
  - – loopback mount (or FUSE when available) needed
- **linspire's click and run (CNR):**
  - – not available at the time
- **RPM/DEB:**
  - \+ Actively supported, users familiar with them, large SW base
  - \+ RPM natural choice for RHEL4 systems: system-wide installed SW available *de facto*
  - – not designed to support concurrently system-wide + local user SW installations

# Examples: bootstrap and DB resynch

- Once: setup the base for the ISOC environment
- download isoc_bootstrap.tgz (or `cvs co`)
- `sh ./scripts/isoc_bootstrap.sh /path/to/rpmstuff/ \`
  `/path/to/installdest`
- Regularly: Synchronize local DB with global install/updates:

```
isoc isoc_resynchdb
  Resynchronizing RPM DB with master
  # DRY RUN mode requested
  Reconciliating changes...
  1 packages to unregister
     1/   1: Unregistering package fetchmail (198): Superseded
  1 packages to register
     1/   1: Registering package fetchmail (1178): Updated
  Do you agree (y/n) ? y
  Reconciliating changes...
  1 packages to unregister
  1/1 packages successfully unregistered
  1 packages to register
  1/1 packages successfully registered
```

# Pitfalls

- Local RPM DB is smaller than system-wide DB but still big !
- Contains the list of files from the global DB (needed for dependency resolution). With it: 20MB, without it: about 1MB...
- Smaller than global DB (3 to 4x) because non-needed infos are discarded (pre/post scripts, ...)
- Initial import takes a loooong time (full install on AFS = 35mn)
- Simple '`cp`' not possible: must tag each packet with their origin for later DB synchronization
- allows to slightly strip down the size of the imported DB (see above)