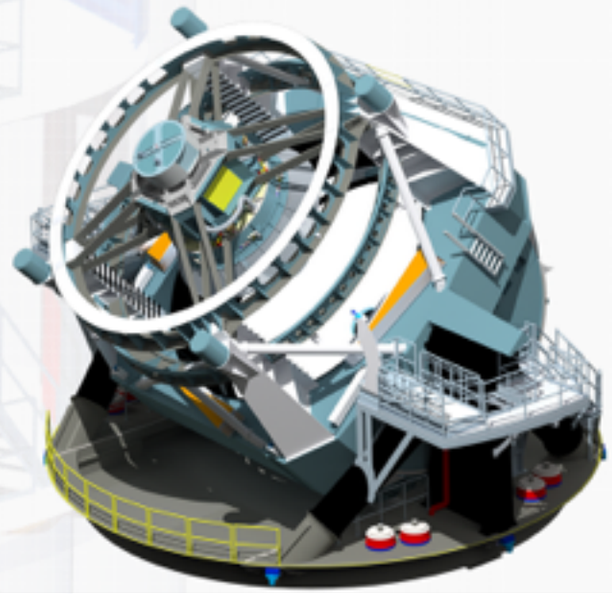
Basic EUPS concepts

John Swinbank

Technical Manager, Data Release Production

5 October 2015



DM Boot Camp
5–7 October 2015 | Princeton

The Most Important Bit

- "Extended Unix Product Support"
  - ExtUPS, EUPS, eups.
- Pronunciation:
  - "Eee-you-pea-ess";
  - Or "ups";
  - **"Ee-ups" in this talk.**

What is it?

- A tool for managing multiple versions of interdependent software packages.
- More concretely:
  - The LSST stack has lot of separate packages (`lsst_apps` depends on 50+; other top level packages bring in more).
  - The relationships between them can be complex (`lsst_apps` version X depends on `afw` version Y which depends on `boost` version at least Z but no greater than Z+N).
  - It's convenient to keep multiple versions of the packages installed at once, for both development and science.
    - Develop targeting today's version of the stack, fix bugs in yesterday's, reproduce your science run from last year…
- EUPS manipulates your environment to make this ~~easy~~ tractable.

Getting started

- Assuming you've installed the LSST stack, you already have EUPS installed.
- But: it's pointing to a big database of all the LSST stack packages, which is hard to get to grips with.
- We can retarget it by setting the ${EUPS_PATH} environment variable.

```
$ . Projects/Astronomy/LSST/stack/loadLSST.bash
$ eups path
/Users/jds/Projects/Astronomy/LSST/stack
$ unsetup lsst
$ export EUPS_PATH="/tmp/eups_demo"
$ mkdir -p ${EUPS_PATH}/ups_db
$ eups path
/tmp/eups_demo
```

- We now have a nice clean database ready to experiment with.

Walk through these commands.

Getting started

- Assuming you've installed the LSST stack, you already have EUPS installed.
- But: it's pointing to a big database of all the LSST stack packages, which is hard to get to grips with.
- We can retarget it by setting the ${ variable.

Sometimes "product" in EUPS-ese; I'll use them interchangeably

```
$ . Projects/Astronomy/LSST/stack/loadLSST.bash
$ eups path
/Users/jds/Projects/Astronomy/LSST/stack
$ unsetup lsst
$ export EUPS_PATH="/tmp/eups_demo"
$ mkdir -p ${EUPS_PATH}/ups_db
$ eups path
/tmp/eups_demo
```

- We now have a nice clean database ready to experiment with.

Walk through these commands.

Getting started

- Assuming you've installed the LSST stack, you already have EUPS installed.
- But: it's pointing [illegible] of all the LSST stack packages, which is [illegible]
- We can [illegible] ATH} environment variable.

Note general form of commands "eups <verb> <options>" (similar to `git`, etc)

```
$ . Projects/A[illegible]omy/LSST/stack/loadLSST.bash
$ eups path
/Users/jds/Projects/Astronomy/LSST/stack
$ unsetup lsst
$ export EUPS_PATH="/tmp/eups_demo"
$ mkdir -p ${EUPS_PATH}/ups_db
$ eups path
/tmp/eups_demo
```

- We now have a nice clean database ready to experiment with.

Walk through these commands.

Getting started

- Assuming you've installed the LSST stack, you already have EUPS installed.
- But: it's pointing to a big database of all the LSST stack packages, which is hard to get to grips with.
- We can retarget it by setting the ${EUPS_PATH} environment variable.

```
$ . Projects/Astronomy/L
$ eups path
/Users/jds/Projects/As
$ unsetup lsst
$ export EUPS_PATH="/tmp/eups_demo"
$ mkdir -p ${EUPS_PATH}/ups_db
$ eups path
/tmp/eups_demo
```

But (un)setup is special!

- We now have a nice clean database ready to experiment with.

Walk through these commands.

Getting started

- Assuming you've installed the LSST stack, you already have EUPS installed.
- But: it's pointing to a big database of all the LSST stack packages, which is hard to get to grips with.
- We can retarget it by setting the ${EUPS_PATH} environment variable.

```
$ . Projects/Astronomy/LSST/stack/loadLSST.bash
$ eups path
/Users/jds/Projects/Astronomy/LSST/stack
$ unsetup lsst
$ export EUPS_PATH="/tmp/eups_demo"
$ mkdir -p ${EUPS_PATH}/ups_db
$ eups path
/tmp/eups_demo
```

Location of the EUPS database

- We now have a nice clean database ready to experiment with.

Walk through these commands.

My contribution to the LSST software effort

- I have written a package I want to use with EUPS.
  - I call it `a`.
  - It's pretty advanced stuff:

```
$ cat python/a.py
__VERSION__ = 1

$ cat bin/a
#!/usr/bin/env python

import a

if __name__ == "__main__":
    print("Package a with vesion %d" % (a.__VERSION__,))
```

  - I've put this in `${EUPS_PATH}/a/v1`.

Describing the package to EUPS

- In order to use this, we want to ensure:
  - That `a.py` is on `${PYTHONPATH}` (so we can import it);
  - That `bin/a` is on `${PATH}` (so we can execute it).
- We communicate this to EUPS via a *table file.*
- Normally, this is in `${PACKAGE_DIR}/ups/${PACKAGE}.table`.

```
$ cat ups/a.table
envPrepend(PYTHONPATH ${PRODUCT_DIR}/python)
envPrepend(PATH ${PRODUCT_DIR}/bin)
```

- We'll meet some other commands which can go in the table file shortly.

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

Standard command form

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

Name of package

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

Version

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

"Root" directory

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

- We can ask EUPS to tell us about all the packages it knows about:

```
$ eups list
a                    v1                  current
```

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

- We can ask EUPS to tell us about all the packages it knows about:

```
$ eups list
a                    v1                    current
```

Name of package

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

- We can ask EUPS to tell us about all the packages it knows about:

```
$ eups list
a                    v1                    current
```

Version

Telling EUPS the package exists

- We can now *declare* the package to EUPS.
  - This records all the information about it in the EUPS database.

```
$ eups declare a v1 -r ${EUPS_PATH}/a/v1
```

- We can ask EUPS to tell us about all the packages it knows about:

```
$ eups list
a                    v1                    current
```

Tag (…coming soon)

Setting it up & tearing it down

```
$ setup a
$ echo $PATH
/tmp/eups_demo/a/v1/bin:…
$ a
Package a with version 1
$ eups list -s
a                    v1                  current setup
$ unsetup a
$ a
-bash: a: command not found
```

Setting it up & tearing it down

${PATH} has been updated correctly

```
$ setup a
$ echo $PATH
/tmp/eups_demo/a/v1/bin:…
$ a
Package a with version 1
$ eups list -s
a                    v1                  current setup
$ unsetup a
$ a
-bash: a: command not found
```

Setting it up & tearing it down

```
$ setup a
$ echo $PATH
/tmp/eups_demo/
$ a
Package a with v
$ eups list -s
a                    v1                  current setup
$ unsetup a
$ a
-bash: a: command not found
```

Only list things currently set-up

Versioning

- Version 2 of our package:
  - Copy ${EUPS_PATH}/a/v1 to ${EUPS_PATH}/a/v2;
  - Bump the version number in a.py;
  - Declare our new version.

```
$ eups declare a v2 -r ${EUPS_PATH}/a/v2
$ eups list
a                    v1                 current
a                    v2
$ setup a
$ a
Package a with version 1
$ unsetup a
$ setup a v2
$ a
Package a with version 2
```

Versioning

- Version 2 of our package:
  - Copy ${EUPS_PATH}/a/v1 to ${EUPS_PATH}/a/v2;
  - Bump the version number in a.py;
  - Declare our new version.

```
$ eups declare a v2 -r ${EUPS_PATH}/a/v2
$ eups list
a                    v1                 current
a                    v2
$ setup a
$ a
Package a with version 1
$ unsetup a
$ setup a v2
$ a
Package a with version 2
```

Getting v1 by default

Versioning

- Version 2 of our package:
  - Copy ${EUPS_PATH}/a/v1 to ${EUPS_PATH}/a/v2;
  - Bump the version number in a.py;
  - Declare our new version.

```
$ eups declare a v2 -r ${EUPS_PATH}/a/v2
$ eups list
a                    v1                 current
a                    v2
$ setup a
$ a
Package a with version 1
$ unsetup a
$ setup a v2
$ a
Package a with version 2
```

Can be explicit about version

Tags

- A name associated with a (set of) version(s).
- There are some standard tags defined by default:

```
$ eups tags
current latest stable user:jds
```

- `current`
  - If you don't do anything "clever", you'll get the version tagged current when you set up a product.
- `latest`
  - Reserved for management(!)
- `stable`
  - You can apply this tag at will; you might find it semantically meaningful.
- `user:${username}`
  - Personal tag; apply at will. Omit the "user".

Note that user tags are defined in your home directory (~/.eups), others are in the eups database. Thus, if other users have access to the database, they can see the global tags but not your local ones.

Also it is possible to define your own free-for tags if you're keen, but you need to edit your EUPS startup file to enable them. We're not going to cover that here: read the fine manual for details.

Applying and using tags

- Apply tags by declaring them:

```
$ eups list
a                    v1                 current
a                    v2
$ eups declare -t stable a v1
$ eups declare -t jds a v2
$ eups list
a                    v1                 current stable
a                    v2                 jds
```

- And use them with (un)setup:

```
$ setup -t jds a
$ a
Package a with version 2
$ setup a
$ a
Package a with version 1
```

Applying and using tags

- Apply tags by declaring them:

```
$ eups list
a                    v1                 current
a                    v2
$ eups declare -t stable a v1
$ eups declare -t jds a v2
$ eups list
a                    v1                 current stable
a                    v2                 jds
```

- And use them with (un)setup:

```
$ setup -t jds a
$ a
Package a with version 2
$ setup a
$ a
Package a with version 1
```

With no tag or version, we get current

Another package

```
$ cat bin/b
#!/usr/bin/env python

import a

if __name__ == "__main__":
    print("Package b is using a version %d" % (a.__VERSION__,))

$ cat ups/b.table
setupRequired(a)
envPrepend(PATH, ${PRODUCT_DIR}/bin)

$ eups declare b v1 -r ${EUPS_PATH}/b/v1

$ eups list
a                    v1           current
a                    v2
b                    v1           current

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: la                             Flavor: Darwin X86  Version: v1

$ b
Package b is using a version 1
```

Another package

We need a to be able to run b

```
$ cat bin/b
#!/usr/bin/e

import a

if __name__ == "__main__":
    print("Package b is using a version %d" % (a.__VERSION__,))

$ cat ups/b.table
setupRequired(a)
envPrepend(PATH, ${PRODUCT_DIR}/bin)

$ eups declare b v1 -r ${EUPS_PATH}/b/v1

$ eups list
a                    v1            current
a                    v2
b                    v1            current

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: la                             Flavor: Darwin X86  Version: v1

$ b
Package b is using a version 1
```

Another package

```
$ cat bin/b
#!/usr/bin/env python

import a

if __name__ == "__main
    print("Package b                                    ERSION__,))

$ cat ups/b.table
setupRequired(a)
envPrepend(PATH, ${PRODUCT_DIR}/bin)

$ eups declare b v1 -r ${EUPS_PATH}/b/v1

$ eups list
a                    v1           current
a                    v2
b                    v1           current

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: la                             Flavor: Darwin X86  Version: v1

$ b
Package b is using a version 1
```

State that explicitly in the table

Another package

```
$ cat bin/b
#!/usr/bin/env python

import a

if __name__ == "__main__":
    print("Package b is using a version %d" % (a.__VERSION__,))

$ cat ups/b.table
setupRequired(a)
envPrepend(PATH, ${PRODUCT_DIR}/bin)

$ eups declare b v1 -r ${EUPS_PATH}/b/v1

$ eups list
a
a
b

$ setup -v b
Setting up: b                                Flavor: Darwin X86  Version: v1
Setting up: la                               Flavor: Darwin X86  Version: v1

$ b
Package b is using a version 1
```

-v: be verbose

Versioned dependencies

- In this case, it doesn't matter which version of *a* is available; *b* is happy with either.

```
$ eups declare -c a v2

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: la                             Flavor: Darwin X86  Version: v2

$ b
Package b is using a version 2
```

Versioned dependencies

Shorthand for "`-t current`"

- In this ca ter which version of *a* is available; *b* is happy with

```
$ eups declare -c a v2

$ setup -v b
Setting up: b                                   Flavor: Darwin X86  Version: v1
Setting up: la                                  Flavor: Darwin X86  Version: v2

$ b
Package b is using a version 2
```

Versioned dependencies

- In this case, it doesn't matter which version of *a* is available; *b* is happy with either.

```
$ eups declare -c a v2

$ setup -v b
Setting up: b                                Flavor: Darwin X86  Version: v1
Setting up: la                               Flavor: Darwin X86  Version: v2

$ b
Package b is using a version 2
```

Always get the current tag

Versioned dependencies

- In this case, it doesn't matter which version of *a* is available; *b* is happy with either.

```
$ eups declare -c a v2

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: |a                             Flavor: Darwin X86  Version: v2

$ b
Package b is using a version 2
```

- Sometimes, we need to be specific:

```
$ cat ups/b.table
setupRequired(a v1)
envPrepend(PATH, ${PRODUCT_DIR}/bin)

$ setup -v b
Setting up: b                              Flavor: Darwin X86  Version: v1
Setting up: |a                             Flavor: Darwin X86  Version: v1
```

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during the build.

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequired(a               -j v2)
} else {
    setupRequired(a v2 [>= v2])
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during t

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequired(a              -j v2)
} else {
    setupRequired(a v2 [>= v2])
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

Run as part of the build

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during the build.

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequire                    -j v2)
} else {
    setupRequ
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

Momentarily…

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during the build.

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequired(a               -j v2)
} else {
    setupRequired(a v2 [>= v2])
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

Just set up this package, not its dependencies

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during the build.

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequired(a               -j v2)
} else {
    setupRequired(a v2 [>= v2])
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

- Pass the `--exact` flag to `setup` to get exactly the packages that were set up when expandtable was run and no more.

```
$ eups list
a                    v1
a                    v3
b                    v1                 current

$ setup --exact b
setup: in file /tmp/eups_demo/b/v1/ups/b.table: Product a v2 not found

$ setup -v --inexact b
Setting up: b                              Flavor: Darwin X86  Version v1
Setting up: la                             Flavor: Darwin X86  Version v3
```

Versioning requirements

- Sometimes (e.g. for ABI compatibility), you really need to record the exact versions used during the build.

```
$ eups expandtable ups/b.table
if (type == exact) {
   setupRequired(a              -j v2)
} else {
    setupRequired(a v2 [>= v2])
}
envPrepend(PATH, ${PRODUCT_DIR}/bin)
```

- Pass the `--exact` flag to `setup` to get exactly the packages that were set up when expandtable was run and no more.

```
$ eups list
a                    v1
a                    v3
b                    v1                 current

$ setup --exact b
setup: in file /tmp/eups_demo/b/v1/ups/b.table: Product a v2 not found

$ setup -v --inexact b
Setting up: b                              Flavor: Darwin X86  Version v1
Setting up: la                             Flavor: Darwin X86  Version v3
```

a v2 has been removed

But which version will we *really* get?

- `current` unless we are “clever”. So what’s clever?
- EUPS tries to find an appropriate version using the “version resolution order”.
  - Analogous to Python’s MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you’re keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

But which version will we *really* get?

- `current` unless we are "clever". So what's clever?
- EUPS tries to find an appropriate version using the "version resolution order".
  - Analogous to Python's MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you're keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

By default, we use exact mode

But which version will we *really* get?

- `current` unless we are "clever". So what's clever?
- EUPS tries to find an appropriate version using the "version resolution order".
  - Analogous to Python's MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you're keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

Use the version specified on the command line

But which version will we *really* get?

- `current` unless we are "clever". So what's clever?
- EUPS tries to find an appropriate version using the "version resolution order".
  - Analogous to Python's MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you're keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

Use an explicit version specified elsewhere (e.g. table says v2)

But which version will we *really* get?

- `current` unless we are "clever". So what's clever?
- EUPS tries to find an appropriate version using the "version resolution order".
  - Analogous to Python's MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you're keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

Use a version expression
(e.g. `>= v2.0`)

But which version will we *really* get?

- `current` unless we are "clever". So what's clever?
- EUPS tries to find an appropriate version using the "version resolution order".
  - Analogous to Python's MRO.
  - Complex because it mixes different types of directive.
  - Can be customized if you're keen…!
  - Default:

```
$ eups vro
type:exact commandLine version versionExpr current
```

Use the `current` tag

Back to LSST

– We're now in a good position to understand the LSST stack:

```
$ . Projects/Astronomy/LSST/stack/loadLSST.bash

$ eups list
activemqcpp          10.1             2015_05 b1327 b1326 […]
[…]

$ eups list -s
doxygen              1.8.5            sims b824 b1630 b1539 […]
python               0.0.2            b261 b182 v9_2-rc1 […]
scons                2.3.0+1          b261 b182 b449 […]
sconsUtils           9.0+2            v9_2 b173 b176 […]
[…]

$ setup -v lsst_apps
Setting up: lsst_apps                     Flavor: DarwinX86  Version: 11.0+3
Setting up: |meas_deblender               Flavor: DarwinX86  Version: 11.0+3
[…]

$ more ${LSST_APPS_DIR}/ups/lsst_apps.table
if (type == exact) {
   setupRequired(meas_deblender  -j 11.0+3)
   setupRequired(utils           -j 11.0-1-g47edd16)
[…]
```

Back to LSST

– We're now in a good position to understand the LSST stack:

```
$ . Projects/Astronomy/LSST/stack/loadLSST.bash

$ eups list
activemqcpp          10.1             2015_05 b1327 b1326 […]
[…]

$ eups list -s
doxygen              1.8.5            sims b824 b1630 b1539 […]
python               0.0.2            b261 b182 v9_2-rc1 […]
scons                2.3.0+1          b261 b182 b449 […]
sconsUtils           9.0+2            v9_2 b173 b176 […]
[…]

$ setup -v lsst_apps
Setting up: lsst_apps                     Flavor: D
Setting up: |meas_deblender               Flavor: D
[…]

$ more ${LSST_APPS_DIR}/ups/lsst_apps.table
if (type == exact) {
   setupRequired(meas_deblender  -j 11.0+3)
   setupRequired(utils           -j 11.0-1-g47edd16)
[…]
```

Lots of tags, corresponding to CI runs, official releases, etc

eups distrib

- A package distribution mechanism & convenient(ish) way to install and update the LSST stack.
- **Distinct** from “core EUPS”, but closely integrated.
- Fetch and install packages defined on some remote server.

```
$ eups distrib path
http://sw.lsstcorp.org/eupspkg

$ eups distrib list lsst_apps
  lsst_apps            generic    8.0.0.1+2
  lsst_apps            generic    8.0.0.1+3
[…]

$ eups distrib install -t v11_0 lsst_apps
```

- To see available tags: https://sw.lsstcorp.org/eupspkg/tags/

eups distrib

- A package distribution mechanism & convenient(ish) way to install and update the LSST stack.
- **Distinct** from "cor… …tegrated.
- Fetch and install p… …he remote server.

This is the LSST distribution server

```
$ eups distrib path
http://sw.lsstcorp.org/eupspkg

$ eups distrib list lsst_apps
  lsst_apps            generic    8.0.0.1+2
  lsst_apps            generic    8.0.0.1+3
[…]

$ eups distrib install -t v11_0 lsst_apps
```

- To see available tags: https://sw.lsstcorp.org/eupspkg/tags/

eups distrib

- A package distribution mechanism & convenient(ish) way to install and update the LSST stack.
- **Distinct** from “core EUPS”, but closely integrated.
- Fetch and install packages defined on some remote server.

```
$ eups distrib path
http://sw.lsstcorp.org/eupspkg

$ eups distrib list lsst_apps
  lsst_apps            generic    8.0.0.1+2
  lsst_apps            generic    8.0.0.1+3
[…]

                       v11_0 lsst_apps
```

Not specific to particular platform

- To see available tags: https://sw.lsstcorp.org/eupspkg/tags/

eups distrib

- A package distribution mechanism & convenient(ish) way to install and update the LSST stack.
- **Distinct** from "core EUPS", but closely integrated.
- Fetch and install packages defined on some remote server.

```
$ eups distrib path
http://sw.lsstcorp.org/eupspkg

$ eups distrib list lsst_apps
  lsst_apps            generic    8.0.0.1+2
  lsst_apps            generic    8.0.0.1+3
[…]

$ eups distrib install -t v11_0 lsst_apps
```

Versions; tags not shown

- To see available tags: https://sw.lsstcorp.org/eupspkg/tags/

For more information

- EUPS is developed outside the LSST stack: it is effectively a third party package.
- Development is managed through the GitHub repository:
  - https://github.com/RobertLuptonTheGood/eups
  - GitHub issue tracker for problems with EUPS itself…
  - …but do report stack installation problems on JIRA.
- There's a (slightly idiosyncratic) LaTeX manual in the repository.
- Some useful tips on the old Trac system:
  - https://dev.lsstcorp.org/trac/wiki/EupsTips