Pybind11 and the LSST Stack

Krzysztof Findeisen

LSST Project Structure With Pybind11

- `include/lsst/project`
  - `Foo.h` — *C++ interface*
- `src/lsst/project`
  - `Foo.cc`
- `python/lsst/project`
  - `foo.cc` — *Python interface (module `_foo`)*
  - `SConscript` — *Compile pybind11 wrappers*
  - `projectLib.py` — *Import wrappers*
  - `__init__.py` — *Combine `projectLib` and pure-Python modules*

LSST Project Structure With Pybind11

- *.cc files independent -- depend only on standard C++ include files
- Order matters in *Lib.py
  - If derived class module imported before base, Pybind11 will raise ImportError
  - May need to import types from other packages
- C++/pybind details should not be present in __init__.py

From Swig to Pybind11

- For “simple” classes, Pybind11 requires more wrapping code, but doesn’t substantially change the Python interfaces
- For more advanced classes, Pybind11 requires fewer workarounds than Swig, so Python interfaces cleaner and (often) more Pythonic
- Everything either pure C++ or pure Python -- no mixing in the same file, and no custom scripting

Run-Time Typing

With Swig:

```
>>> c = afwCoord.makeCoord(afwCoord.FK5)
>>> type(c)
<class 'lsst.afw.coord.coordLib.Coord'>
>>> myCoord = afwCoord.Fk5Coord.cast(c)
>>> type(myCoord)
<class 'lsst.afw.coord.coordLib.Fk5Coord'>
```

With Pybind11:

```
>>> c = afwCoord.makeCoord(afwCoord.FK5)
>>> type(c)
<class 'lsst.afw.coord._coord.Fk5Coord'>
```

Better Template/STL Support

```
%template(FlagDefinitionVector) vector<FlagDefinition>;
```

```
flagDefs = [FlagDefinition("General", "general failure"),
            FlagDefinition("1st", "first failure type"),
            FlagDefinition("2nd", "second failure type")
            ]
fh = FlagHandler.addFields(schema, "test",
                           FlagDefinitionVector(flagDefs))
```

Better Template/STL Support

```
clsFlagHandler.def_static("addFields",
    (FlagHandler (*)(Schema &, string const &,
        vector<FlagDefinition> const *)) &FlagHandler::addFields);
```

```
flagDefs = [FlagDefinition("General", "general failure"),
            FlagDefinition("1st", "first failure type"),
            FlagDefinition("2nd", "second failure type")
            ]
fh = FlagHandler.addFields(schema, "test", flagDefs)
```

Cleaner Patching

```
%extend Extent<T,N> {
PyObject * __iadd__(PyObject** PYTHON_SELF, Extent<T,N> const & other) {
    *self += other;
    Py_INCREF(*PYTHON_SELF);
    return *PYTHON_SELF;
}
}


cls.def("__iadd__", [](Extent<T,N> & self, Extent<T,N> const & other) {
    return self += other;
});
```