Linking External Software to the ORIGEN API

2018 SCALE Users' Group Tutorial

W. Wieselquist
S. Hart
K. Dugan

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Who uses the ORIGEN API?

- CASL VERA-CS core simulator
- ADVANTG with activation mode
- IN-DEPTH
- SCALE TRITON/Polaris
- ORIGEN sequence itself
- CYBORG (Cyclus fuel cycle simulator with ORIGEN)
- ORION fuel cycle simulator

Dependencies

- SCALE 6.2
  - LAPACK
  - TRILINOS
  - QT

- SCALE 6.3
  - LAPACK
  - ~~TRILINOS~~
  - ~~QT~~
  - HDF5

Capabilities

- Anything that ORIGEN can do in SCALE!
  - Decay/irradiation/activation
  - Unit conversions
  - Emission calculations
  - Binary ORIGEN concentration file (F71) manipulation
  - Etc.

Layout

- Some knowledge of CMake/TriBITs configuration system is helpful
- Main package: $SCALE/packages/Origen
- Supackages:
  - Core (lowest)
  - Solver (depends on Core)
  - Manager (depends on Solver)

Layout (cont.)

- Supackages:
  - Core (lowest)
    - dc - data containers
    - io - input/output routines
    - re - data resources
    - ts - transition system
    - fn - function library
    - ut - utility executables
    - xf - interfaces
  - Solver (depends on Core)
  - Manager (depends on Solver)

ORIGEN API story

- Evolved from need to modernize/modularize *and* simultaneously integrate with other codes
  - Lots of existing capability which slowly morphed
  - Multiple authors
  - Started in Fortran, ended in C++ (C++ with auto bindings moving forward)
  - *Is messy! More than one way to do something (old way and new way with new way not 100% adopted)*
- Want to do both
  - Maximize code reuse within SCALE
  - Limit dependencies on other packages in SCALE

ORIGEN API story (cont.)

- Some bad design
  - Class packages/Origen/Core/dc/**TransitionMatrixP** tries to do some things it cannot reliably and has horrible naming, get/set paradigm with no consistency checking
  - Class packages/Origen/Core/dc/**Material** has too many accessors
- Some good design
  - Class packages/Origen/Core/ts/* is a cohesive set of classes for storing/updating/accessing transition properties
  - Interface packages/Origen/Core/xf/**Solver** is a light solver interface (unifies old "MATREX" solver and "CRAM" solver)

ORIGEN API story (cont.)

- Priorities for sponsors
  - Fast, accurate, repeatable, …
  - Allowed us to neglect documentation of API ☹
    - Best documentation is the usage of the code in unit tests and throughout code base--need source license for that
    - A github site hosts documentation https://wawiesel.github.io/OrigenAPI-Demo/dd/d60/tst_material_8cpp-example.html
- As we gain confidence with C++ best practices and figure out the right classes, we will crystallize API
  - Minimal set of C++ classes with Python, Fortran bindings through SWIG
  - Gradual deprecation of "extra" classes

Link your "app" to ORIGEN

Method	Pros	Cons
1. Link "app" against standard SCALE install from **RSICC**	• Do not have to build SCALE • Works with executable-only version • Do not have to install third-party libraries (TPLs)	• Only works with C++ API • Must use compatible compiler (see SCALE README) • Need to develop linking commands (dependent libraries)
2. Link "app" against **your build** of SCALE	• Compiler compatibility • Can use Fortran API	• Must install TPLs • Takes time to build SCALE (but one time cost) • Need to develop linking commands (dependent libraries)
3. Build "app" with SCALE	• Compiler compatibility • Can use Fortran API • Least likely to break with SCALE changes	• Must install TPLs • Takes time to build SCALE • Need to learn a little CMake/TriBITS

Goals

- Use the C++ API from a standard RSICC Linux install
- Create a library and link an executable
- Get comfortable searching ORIGEN
  - source tree
  - docs online (https://wawiesel.github.io/OrigenAPI-Demo/)
- Learn where ORIGEN tests are

Exercises

1. Load an ORIGEN reaction resource and output some details (`rr_output.cpp`)
   - `Core/re/ReactionResource.h`
   - `Core/io/ReactionResourceIO.h`
   - `Core/io/tstReactionResourceIO.cpp`

2. Create an ORIGEN concentration file and view in Fulcrum (`myf71.cpp`)
   - `Core/dc/StateSet.h`
   - `Core/dc/Concentrations.h`

3. Solve a decay problem using the Material API (`decaythis.cpp`)
   - `Core/dc/Material.h`
   - `Solver/SolverSelector.h`

Exercises (cont.)

- We have created a simple CMake project for each exercise to execute method #1: Link "app" against standard SCALE install from RSICC.
- Each project should link and compile.

Method	Pros	Cons
1. Link "app" against standard SCALE install from **RSICC**	• Do not have to build SCALE • Works with executable-only version • Do not have to install third-party libraries (TPLs)	• Only works with C++ API • Must use compatible compiler (see SCALE README) • Need to develop linking commands (dependent libraries)
2. Link "app" against **your build** of SCALE	• Compiler compatibility • Can use Fortran API	• Must install TPLs • Takes time to build SCALE (but one time cost) • Need to develop linking commands (dependent libraries)
3. Build "app" with SCALE	• Compiler compatibility • Can use Fortran API • Least likely to break with SCALE changes	• Must install TPLs • Takes time to build SCALE • Need to learn a little CMake/TriBITS

Summary

- We hope you got a "feel" for the API today
  - Hardest part is linking
  - Next hardest part is knowing which part of the API to use
- Our plan
  - **Origen::Material** will become the main entry point with dependency on a few other interfaces (Solver, Library, TransitionMatrix)
  - Continue to extend **Resources** to have more input/output file formats
  - **Origen::Concentrations** is the main storehouse for isotopic results
  - Emission calcs and unit conversions are actions on **Concentrations**