ECLIPSE Compatible Restart Files for Flow

Structure, Output and Input

Bård Skaflestad

SINTEF Digital, Mathematics & Cybernetics

TNO, Utrecht, 24 January 2019

E-mail: Bard.Skaflestad@sintef.no

Primary Benefits for OPM/Flow

- Integration into existing workflows at enterprise scale
- Enable combined Flow/ECLIPSE simulation (e.g., use Flow to simulate the history period and ECLIPSE for the prediction part)
- Generate continuous cumulative production values when restarting a simulation run

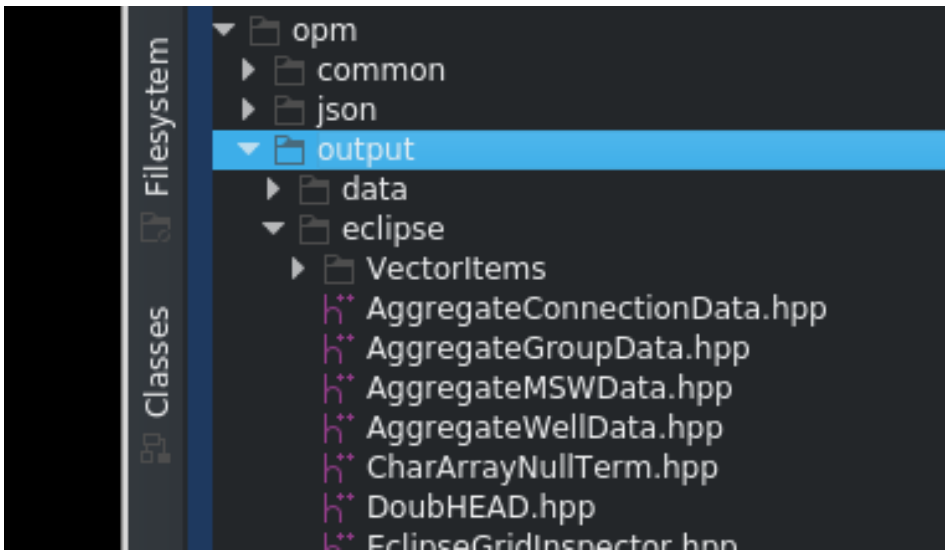


Figure : You are here

Project Overview

Time Line

- Project start November 2017
- First ECLIPSE restart July 2018
- Model 2 restart (ECLIPSE) October 2018 (segment data)
- Flow restart Soon™ (PR #548)

Acknowledgements

- Equinor
- Jostein Alvestad, Torbjørn Skille
- Joakim Hove (`libecl`, existing solution output)

Gateway Functions (`opm/output/eclipse/RestartIO.hpp`)

- `RestartIO::save()` (called from `EclipseIO::writeTimeStep()`)
- `RestartIO::load()` (called from `EclipseIO::loadRestart()`)

Process

1. Grab independent copy of `libecl`'s output code, port `RestartIO::save()`
2. Identify sizes, items, substructures
3. Fill in items, output additional vectors (ongoing)
4. Leverage knowledge to input same items for simulation restart

Basic Structure of Single Restart Block

Basic Constraints on Project

Contents

1. Report step number (unified only)
2. Step headers
3. Well, group, completion, segment arrays (as needed)
4. Solution arrays

Step Headers

- `INTEHEAD` Array sizes, timestamps, unit conventions
- `LOGIHEAD` Flags to enable/disable effects
- `DOUBHEAD` Tuning parameters, timestamp, next timestep

Well, Group, Segment Arrays

Constraints (targets and limits), number of reservoir connections, group tree topology, instantaneous rates, potential rates, cumulative totals etc.

Solution Arrays

Pressure, saturations, composition (R_s and R_v), hysteresis variables (maximum saturations etc.)

Fundamental Data Types

In Restart File

Arrays of (FORTRAN)

- `Integer` (`int`)
- `Real` (`float`)
- `Double Precision` (`double`)
- `Character*8` (`char[9]`)

Normalised Simulator Objects

- `RestartValue`
- `SummaryState`
- `Schedule`, `EclipseState`
- `EclipseGrid`

`RestartValue`

```
class RestartValue {
public:
    data::Solution solution;
    data::Wells wells;
    // ...
};
```

`SummaryState`

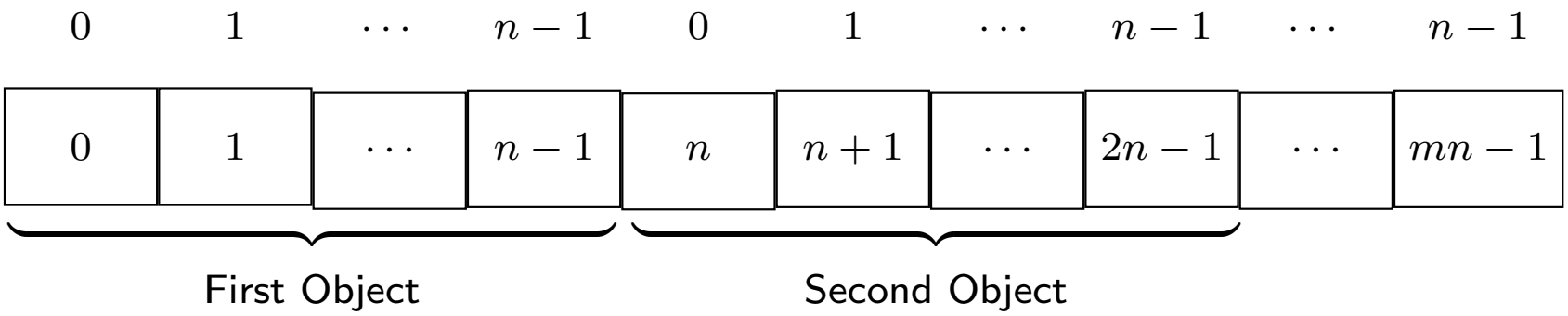
Collection of Well-, Group-, Field-, Connection-, Segment-

- Flow rates
- Cumulative totals
- Potential rates
- Water cut...

Restart Code's Primary Objective

Output `RestartValue` and `SummaryState` as FORTRAN arrays in a manner that enables high fidelity reconstruction of the objects on simulation restart.

Object Array Substructure



Windowed Array

Helper class that knows about the number of items n and the number of objects m that collectively make up the complete linearised object array. Provides read/write access to the array elements of a single object (well, group, segment etc.).

Windowed Matrix

Extends Windowed Array to the case of m objects having k sub-objects, each of which has n items of information in the complete linearised object array. Mainly intended for data items associated to a well's reservoir connections.

Direct Implication

Object arrays are *typically* sized according to the various **maximum** sizes defined in the `RUNSPEC` section. These sizes need to be correct (i.e., "big enough") when using ECLIPSE compatible restart files.

System for Recording Known Vector Items

opm/output/eclipse/VectorItems/*.hpp

```
namespace Opm { namespace RestartIO { namespace Helpers { namespace VectorItems {

    namespace IWell {
        enum index : std::vector<int>::size_type {
            // ...
            NConn    =  4, // Number of active cells connected to well
            Group    =  5, // Index (one-based) of well's current group
            WType    =  6, // Well type (producer vs. injector)
            ActWCtrl =  7, // Well's active target control mode (constraint).
            // ...
        };
    } // IWell

    namespace XWell {
        enum index : std::vector<double>::size_type {
            OilPrRate  =  0, // Well's oil production rate
            WatPrRate  =  1, // Well's water production rate
            GasPrRate  =  2, // Well's gas production rate
            LiqPrRate  =  3, // Well's liquid production rate
            VoidPrRate =  4, // Well's reservoir voidage production rate

            FlowBHP    =  6, // Well's flowing/producing bottom hole pressure
            WatCut     =  7, // Well's producing water cut
            GORatio    =  8, // Well's producing gas/oil ratio

            OilPrTotal   = 18, // Well's total cumulative oil production
            // ...
        };
    } // XWell
}}}} // Opm::RestartIO::Helpers::VectorItems
```

Outputting Well and Connection Data

Aggregate Well Data

- Integer, float, double, and character arrays for all wells
- Distinguishes static/declared items from dynamic data
- Array sizes from `INTEHEAD`
- Wells in order of appearance in `SCHEDULE` within each array

Well Loop

```
template <typename WellOp>
void wellLoop(/* ... */,
              WellOp&& wellOp)
{
    for (all non-null wells) {
        wellOp(*well, wellID);
    }
}
```

Aggregate Connection Data

- Integer, float, double, and character arrays for all connections for all wells
- Does not Distinguish static/declared items from dynamic data
- Array sizes from `INTEHEAD`, allocates maximum number of connections for each well
- Wells in order of appearance in `SCHEDULE` within each array, connections in order of appearance in `COMPDAT`

Connection Loop

Loops all connections for all wells in right order, invokes a connection operation for each connection.

Outputting Well Data (cont'd)

Creating XWEL for a Producer

Well Loop Invocation

```
// Dynamic contributions to XWEL array.
wellLoop(wells, [this, sim_step, ecl_compatible_rst, &smry]
    (const Well& well, const std::size_t wellID) -> void
{
    auto xw = this->xWell_[wellID];

    XWell::dynamicContrib(well, smry, sim_step, ecl_compatible_rst, xw);
});
```

XWell::dynamicContrib()

```
template <class XWellArray>
void assignProducer(const std::string&          well,
                    const ::Opm::SummaryState& smry,
                    const bool  ecl_compatible_rst,
                    XWellArray&                 xWell)
{
    using Ix = ::Opm::RestartIO::Helpers::VectorItems::XWell::index;

    xWell[Ix::OilPrRate] = smry.get("WOPR:" + well);
    // ...
}
```

Outputting Well Data (cont'd)

Helper function `writeWell()`—`RestartIO.cpp`

```
auto wellData = Helpers::AggregateWellData(ih);
wellData.captureDeclaredWellData(/* ... */);
wellData.captureDynamicWellData(/* ... */);

write_kw(rst_file, "IWEL", wellData.getIWell());
write_kw(rst_file, "SWEL", wellData.getSWell());
write_kw(rst_file, "XWEL", wellData.getXWell());
write_kw(rst_file, "ZWEL", serialize_ZWEL(wellData.getZWell()));
```

Helper Function `write_kw()`

```
template <typename T>
void write_kw(::Opm::RestartIO::ecl_rst_file_type* rst_file,
              const std::string&                   keyword,
              const std::vector<T>&                data)
{
    const auto kw = EclKW<T>(keyword, data);

    // Call into libecl to write actual keyword data
    ::Opm::RestartIO::ecl_rst_file_add_kw(rst_file, kw.get());
}
```

Special Considerations for Solution Arrays

Fundamental Restrictions for ECLIPSE Compatibility

- Cannot output arrays that ECLIPSE does not know about (e.g., `OPM_*`)
- Cannot output *known* arrays if the corresponding feature is not active (e.g., `TEMP` in a non-thermal simulation run)
- Cannot output global solution arrays outside of the `STARTSOL`/`ENDSOL` demarcation.

Remedies Implemented During Project

1. Make `OPM_*` vector output contingent on not creating a strictly compatible restart file
2. Output Flow's hysteresis variables in terms of ECLIPSE's `SOMAX` and `SGMAX` variables (sufficient for test asset; will probably need refinement when faced with more involved hysteresis behaviour)
3. Output `TEMP` only when run activates thermal option
4. Don't output `RESTART_AUXILIARY` vectors at all (special case handling of `THRESPR`; moved into solution section)

Restarting From ECLIPSE Compatible Restart File

`src/opm/output/eclipse/LoadRestart.cpp`

RestartFileView

- Helper class that wraps an object of type `ecl_file_view_type` from `libecl`, restricted to keywords pertaining to requested report (restart) step
- Conversion operators to behave like the wrapped object in `libecl` function calls
- Supports retrieving individual keywords from this restricted view

{Well, Group, Segment}Vectors

- Helper classes that know the underlying structure of the connection, well, group, and segment related restart vectors
- Behaves similarly to read-only Windowed Arrays (could possibly be implemented in terms of those)

Basic Structure

1. Construct `RestartFileView` of requested restart step
2. Load solution arrays, translating ECLIPSE hysteresis variables to Flow's variables if needed
3. Load well, connection, group, and segment variables
4. Form `RestartValue` and `SummaryState` (cumulative quantities) objects