Documentation on `pp_ser.py`

Xiaolin Guo

November 22, 2015

v0.1

`pp_ser.py` is a parser to expand `!$SER` serialization directives in Fortran code in order to generate serialization code using the `m_serialize.f90` interface for the STELLA serialization framework.

The grammar is defined by a set of `!$SER` directives. All directives are case-insensitive. The main keywords are `INIT` for initialization, `VERBATIM` for echoing some Fortran statements, `OPTION` for setting specific options for the serialization module, `REGISTER` for registering a data field meta-information, `ZERO` for setting some field to zero, `SAVEPOINT` for registering a savepoint with some optional information, `DATA` for serializing a data field, and `CLEANUP` for finishing serialization.

`pp_ser.py` adds the macro `SERIALIZE` around the translated directives like this:

```
#ifdef SERIALIZE
...
#end
```

`pp_ser.py` also supports serializing data on GPU with OpenACC directives.

1 Keywords

1.1 INIT

Initialize the serialization framework.

```
!$ser init directory=dir prefix=pre [mode=] [prefix_ref=] [if
   if_statement]
```

- `dir`: the directory of the main database
- `prefix`: the prefix of the main database file names

- `mode`: 0 or 1
- `prefix_ref`: the prefix of the reference database file names. If reference database is set, the serializer reads from the reference database and writes to the main database.
- `if_statement`: under which condition is the directive executed

Examples:

```
!$ser init directory='.' prefix='Field'
!$ser init directory='.' prefix='database' prefix_ref='ref_database'
!$ser init directory='.' prefix='Field' if ser_test_mode==0
```

1.2 MODE

Set the serialization mode (`ppser_mode`). 0 = write, 1 = read. `ppser_mode` is by default 0.

```
!$ser mode [[read | write] | [0 | 1] | variable] [if if_statement]
```

Examples:

```
!$ser mode 0
!$ser mode read if i_am_accel_node
!$ser mode ser_test_mode
```

1.3 DATA

Serialize the data field.

```
!$ser data field=variable [field2=variable2 ...] [removeintentin] [
   if if_statement]
```

If `removeintentin` is stated, `pp_ser.py` will find the declarations of all the variables in that directive and remove `INTENT(IN)`:

```
!$ser data pp=pp(:) removeintentin
```

is translated to

```
#ifdef SERIALIZE
  REAL(wp) :: pp(kbdim)
#else
  REAL(wp), INTENT (IN) :: pp(kbdim)
#endif
```

Examples:

```
!$ser data pt=pt(:,kk) pq=pq(:,kk)
!$ser data pp=pp(:) removeintentin
!$ser data pp_field=pp pq=pq if test_counter<30
```

1.4 VERBATIM

Examples:

```
!$ser verbatim PRINT *, 'ser_test_mode =', ser_test_mode
```

See more examples in Section 2.

1.5 SAVEPOINT

Create a savepoint with optional meta information.

```
!$ser savepoint name [meta-information] [if if_statement]
```

Examples:

```
!$ser savepoint cuadjtq.DoStep-in iteration=test_counter
!$ser savepoint cuadjtq_out if i_am_accel_node
```

1.6 ON and OFF

Turn on and turn off the serializer.

```
!$ser [on | off]
```

Examples:

```
!$ser on
!$ser off
```

1.7 ZERO

Set the field to be zero.

```
!$ser zero variable1 [variable2 ...] [if if_statement]
```

Examples:

```
!$ser zero test_counter
!$ser zero pt pq
!$ser zero pt if test_counter==0
```

2 Examples

2.1 A first example

Here is a first example of how the directives look like in real codes.

```
MODULE mo_cuadjust
...
SUBROUTINE cuadjtq()
ACC_PREFIX DATA PCOPY(pt(:,kk), pq(:,kk)), IF (i_am_accel_node)

    !$ser init directory='.' prefix='Field'

    !$ser savepoint cuadjtq.DoStep-in
    !$ser mode write
    !$ser data pt=pt(:,kk) pq=pq(:,kk)
    !$ser data pp=pp(:) removeintentin

ACC_PREFIX PARALLEL, IF (i_am_accel_node)
...
ACC_PREFIX END PARALLEL

    !$ser savepoint cuadjtq.DoStep-out
    !$ser mode write
    !$ser data pt=pt(:,kk) pq=pq(:,kk)

ACC_PREFIX END DATA
END SUBROUTINE cuadjtq
END module mo_cuadjust
```

In this example, we first initialize the serializer, then create a new savepoint for the inputs and set the mode. We serialize `pt`, `pq` and `pp`. Here `pp` is marked as `INTENT(IN)` and we need to remove that. After some computation, we create a savepoint for the outputs, set the mode and serialize the data again.

2.2 Decide the mode dynamically

What if we need one executable to perform differently: sometimes reading input and sometimes writing the input. This is achieved by

```
MODULE mo_cuadjust
  !$ser verbatim USE mo_run_config, ONLY: ser_test_mode
  IMPLICIT NONE
SUBROUTINE cuadjtq()
    !$ser savepoint cuadjtq.DoStep-in
    !$ser mode ser_test_mode
    !$ser data ...

END SUBROUTINE cuadjtq
END module mo_cuadjust
```

Here `ser_test_mode` is in the namelist `mo_run_config` (added in the namelist manually!). If we want to write the inputs, we set `ser_test_mode` to be 0 in the configuration file. Otherwise we set it to be 1.

2.3 Adding meta-information to a savepoint

Savepoints are considered equal if they have the same name and the same meta-information. If the subroutine in 2.1 is called twice, then we end up creating the same savepoint twice. To solve this, we keep track of how many times the subroutine is called and add this as meta-information.

```
MODULE mo_cuadjust

  IMPLICIT NONE

  PRIVATE

  !$ser verbatim INTEGER :: test_counter = 0    ! number of times
     subroutine is called
SUBROUTINE cuadjtq()

    !$ser savepoint cuadjtq.DoStep-in iteration=test_counter

ACC_PREFIX PARALLEL, IF (i_am_accel_node)
...
ACC_PREFIX END PARALLEL

    !$ser savepoint cuadjtq.DoStep-out iteration=test_counter

    !$ser verbatim test_counter = test_counter + 1
```

```
END SUBROUTINE cuadjtq
END module mo_cuadjust
```

2.4 Turn off the serializer after a certain number of function call

With the counter defined in 2.3, we can turn off the serializer if we do not want it to produce too much data.

```
    !$ser verbatim IF (test_counter<100) THEN
    !$ser off
    !$ser verbatim ENDIF
```

We can also set the serializer to be off at the beginning, and turn it on from the point we feel like serializing.

```
    !$ser off
    !$ser verbatim IF (test_counter>10) THEN
    !$ser on
    !$ser verbatim ENDIF
```

2.5 Reference database

An extension to 2.2 is that when we write the input, we are generating the reference database. When we read the input, we are generating the test database, and we would like to read from the reference database and write to the test database. One solution to this is

```
MODULE mo_cuadjust
  !$ser verbatim USE mo_run_config, ONLY: ser_test_mode
  IMPLICIT NONE
SUBROUTINE cuadjtq()
    !$ser init directory='.' prefix='ref_' if ser_test_mode==0
    !$ser init directory='.' prefix='test_' prefix_ref='ref_' if
       ser_test_mode==1

    !$ser savepoint cuadjtq.DoStep-in
    !$ser mode ser_test_mode
    !$ser data ...

END SUBROUTINE cuadjtq
END module mo_cuadjust
```

If `ser_test_mode` is 0, we are generating the reference database. If it is 1, we are generating the test database.

2.6 Multiple MPI nodes

When we are running on multiple MPI nodes, we do not want the serializer to write to the same database. To solve this, we add the MPI rank to the prefix of the database.

```
MODULE mo_cuadjust
  !$ser verbatim USE mo_mpi,        ONLY: get_my_global_mpi_id
  IMPLICIT NONE
SUBROUTINE cuadjtq()
!get mpi rank
    !$ser verbatim mpi_id = get_my_global_mpi_id()
    !$ser verbatim Write( str_prefix, '(I1)' ) mpi_id
    !$ser verbatim str_prefix = 'rank'//trim(str_prefix)//'_field'

!initialize the serialization framework
    !$ser init directory='.' prefix='ref_'//str_prefix

END SUBROUTINE cuadjtq
END module mo_cuadjust
```