National Unified Operational Prediction Capability

NUOPC Layer Reference

ESMF v6.2.0

CSC Committee Members

May 17, 2013
3.9.20 NUOPC_GridCompAreServicesSet ................................................. 37
3.9.21 NUOPC_GridCompAttributeAdd .................................................. 38
3.9.22 NUOPC_GridCompCheckSetClock ................................................. 38
3.9.23 NUOPC_GridCompSetClock .......................................................... 39
3.9.24 NUOPC_GridCreateSimpleXY ......................................................... 39
3.9.25 NUOPC_IsCreated ............................................................ 40
3.9.26 NUOPC_StateAdvertiseField ......................................................... 40
3.9.27 NUOPC_StateBuildStdList .......................................................... 41
3.9.28 NUOPC_StateIsAllConnected ...................................................... 42
3.9.29 NUOPC_StateIsAtTime ................................................................. 42
3.9.30 NUOPC_StateIsFieldConnected .................................................... 42
3.9.31 NUOPC_StateIsUpdated .............................................................. 43
3.9.32 NUOPC_StateRealizeField ........................................................... 43
3.9.33 NUOPC_StateSetTimestamp .......................................................... 44
3.9.34 NUOPC_StateUpdateTime ............................................................. 44
3.9.35 NUOPC_TimePrint ................................................................. 45
1 Description

The NUOPC Layer is an add-on to the standard ESMF library. It consists of generic code of two different kinds: utility routines and generic components. The NUOPC Layer further implements a dictionary for standard field metadata.

The utility routines are subroutines and functions that package frequently used calling sequences of ESMF methods into single calls. Unlike the pure ESMF API, which is very class centric, the utility routines of the NUOPC Layer often implement tasks that involve several ESMF classes.

The generic components are provided in form of Fortran modules that implement GridComp and CplComp specific methods. Generic components are useful when implementing NUOPC compliant driver, model, mediator, or connector components. The provided generic components form a hierarchy that allows the developer to pick and choose the appropriate level of specification for a certain application. Depending on how specific the chosen level, generic components require more or less specialization to result in fully implemented components.

2 Design and Implementation Notes

The NUOPC Layer is implemented in Fortran on top of the public ESMF Fortran API.

The NUOPC utility routines form a very straight forward Fortran API, accessible through the NUOPC Fortran module. The interfaces only use native Fortran types and public ESMF derived types. In order to access the utility API of the NUOPC Layer, user code must include the following two use lines:


cue ESMF
cue NUOPC

2.1 Generic Components

The NUOPC generic components are implemented as a collection of Fortran modules. Each module implements a single, well specified set of standard ESMF_GridComp or ESMF_CplComp methods. The nomenclature of the generic component modules starts with the NUOPC_prefix and continues with the flavor: Driver, Model, Mediator, or Connector. This is optionally followed by a string of additional descriptive terms. The four flavors of generic components implemented by the NUOPC Layer are:

- **NUOPC_Driver** - A generic driver component. It implements a child component harness, made of State and Component objects, that follows the NUOPC Common Model Architecture. It is specialized by plugging Model, Mediator, and Connector components into the harness. Driver components can be plugged into the harness to construct component hierarchies. The generic Driver initializes its child components according to a standard Initialization Phase Definition, and drives their Run() methods according a customizable run sequence.

- **NUOPC_Model** - A generic model component that wraps a model code so it is suitable to be plugged into a generic Driver component.

- **NUOPC_Mediator** - A generic mediator component that wraps custom coupling code (flux calculations, averaging, etc.) so it is suitable to be plugged into a generic Driver component.

- **NUOPC_Connector** - A generic component that implements Field matching based on metadata and executes simple transforms (Regrid and Redist). It can be plugged into a generic Driver component.
The user code accesses the desired generic component(s) by including a use line for each one. Each generic component defines a small set of public names that are made available to the user code through the use statement. At a minimum the SetServices method is made public. Some generic components also define a public internal state type by the standard name InternalState. It is recommended that the following syntax is used when accessing a generic component (here with internal state):

```fortran
use NUOPC_DriverXYZ, only: 
     DriverXYZ_SS => SetServices, 
     DriverXYZ_IS => InternalState
```

A generic component is used by user code to implement a specialized version of the component. The user code therefore also must implement a public SetServices routine. The first thing this routine must do is call into the SetServices routine provided by the generic component. It is through this step that the specialized component inherits from the generic component.

There are three mechanisms through which user code specializes generic components.

1. The specializing user code must set entry points for standard component methods not implemented by the generic component. Methods (and phases) that need to be implemented are clearly documented in the generic component description. The user code may further overwrite standard methods already implemented by the generic component code. However, this should rarely be necessary, and may indicate that there is a better fitting generic component available. Finally, some generic components come with generic routines that are suitable candidates for the standard component methods, yet require that the specializing code registers them as appropriate. Setting entry points for standard component methods is done in the SetServices routine right after calling into the generic SetServices method.

2. Some generic components require that specific methods are attached to the component. If a generic component uses specialization through attachable methods, the specific method labels (i.e. the names by which these methods are registered) and the purpose of the method are clearly documented. In some cases attachable methods are optional. This is clearly documented. Further, some generic components attach a default method to a label, which then is used for all phases. This default can be overwritten with a phase specific attachable method. Attaching methods to the component should be done in the SetServices routine right after setting entry points for the standard component methods.

3. Some generic components provide access to an internal state type. The documentation of a generic component indicates which internal state members are used for specialization, and how they are expected to be set. Setting internal state members often requires the availability of other pieces of information. It may happen in the SetServices routine, but more often inside a specialized standard entry point or an attachable method.

Components that inherit from a generic component may choose to only specialize certain aspects, leaving other aspects unspecified. This allows a hierarchy of generic components to be implemented with a high degree of code re-use. The variable level of specialization supports the very differing user needs. Figure 1 depicts the inheritance structure of the NUOPC Generic Components. There are two trees, one is rooted in ESMF_GridComp, while the other is rooted in ESMF_CplComp.
2.2 Field Dictionary

The NUOPC Layer uses standard metadata on Fields to guide the decision making that is implemented in generic code. The generic NUOPC_Connector component, for instance, uses the StandardName Attribute to construct a list of matching Fields between the import and export States. The NUOPC Field Dictionary provides a software implementation of a controlled vocabulary for the StandardName Attribute. It also associates each registered StandardName with canonical Units, a default LongName, and a default ShortName.

The NUOPC Layer provides a number of default entries in the Field Dictionary, shown in the table below. The StandardName Attribute of all default entries complies with the Climate and Forecast (CF) conventions as documented at [http://cf-pcmdi.llnl.gov/](http://cf-pcmdi.llnl.gov/).

Currently it is typically that a user of the NUOPC Layer extends the Field Dictionary by calling the `NUOPC_FieldDictionaryAddEntry()` interface to add additional entries. It is our intention to grow the number of default entries over time, and to more strongly leverage the NUOPC Field Dictionary to ensure metadata interoperability between codes that use the NUOPC Layer.

Besides the StandardName Attribute, the NUOPC Layer currently only uses the Units entry to verify that Fields are given in their canonical units. The plan is to extend this to support unit conversion in the future. The default LongName and default ShortName associations are provided as a convenience to the implementor of NUOPC compliant components; the NUOPC Layer itself does not base any decisions on these two Attributes.

<table>
<thead>
<tr>
<th>StandardName</th>
<th>Units</th>
<th>LongName</th>
<th>ShortName</th>
</tr>
</thead>
<tbody>
<tr>
<td>air_pressure_at_sea_level</td>
<td>Pa</td>
<td>Air Pressure at Sea Level</td>
<td>pmsl</td>
</tr>
<tr>
<td>Variable</td>
<td>Unit</td>
<td>Description</td>
<td>Package</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------</td>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>magnitude_of_surface_downward_stress</td>
<td>Pa</td>
<td>Magnitude of Surface Downward Stress</td>
<td>taum</td>
</tr>
<tr>
<td>precipitation_flux</td>
<td>kg m(^{-2}) s(^{-1})</td>
<td>Precipitation Flux</td>
<td>prcf</td>
</tr>
<tr>
<td>sea_surface_height_above_sea_level</td>
<td>m</td>
<td>Sea Surface Height Above Sea Level</td>
<td>ssh</td>
</tr>
<tr>
<td>sea_surface_salinity</td>
<td>1e-3</td>
<td>Sea Surface Salinity</td>
<td>sss</td>
</tr>
<tr>
<td>sea_surface_temperature</td>
<td>K</td>
<td>Sea Surface Temperature</td>
<td>sst</td>
</tr>
<tr>
<td>surface_eastward_sea_water_velocity</td>
<td>m s(^{-1})</td>
<td>Surface Eastward Sea Water Velocity</td>
<td>sscu</td>
</tr>
<tr>
<td>surface_downward_eastward_stress</td>
<td>Pa</td>
<td>Surface Downward Eastward Stress</td>
<td>tauu</td>
</tr>
<tr>
<td>surface_downward_heat_flux_in_air</td>
<td>W m(^{-2})</td>
<td>Surface Downward Heat Flux in Air</td>
<td>hfns</td>
</tr>
<tr>
<td>surface_downward_water_flux</td>
<td>kg m(^{-2}) s(^{-1})</td>
<td>Surface Downward Water Flux</td>
<td>wfns</td>
</tr>
<tr>
<td>surface_downward_northward_stress</td>
<td>Pa</td>
<td>Surface Downward Northward Stress</td>
<td>tauv</td>
</tr>
<tr>
<td>surface_net_downward_shortwave_flux</td>
<td>W m(^{-2})</td>
<td>Surface Net Downward Shortwave Flux</td>
<td>rsns</td>
</tr>
<tr>
<td>surface_net_downward_longwave_flux</td>
<td>W m(^{-2})</td>
<td>Surface Net Downward Longwave Flux</td>
<td>rlns</td>
</tr>
<tr>
<td>surface_northward_sea_water_velocity</td>
<td>m s(^{-1})</td>
<td>Surface Northward Sea Water Velocity</td>
<td>sscv</td>
</tr>
</tbody>
</table>

2.3 Metadata

2.3.1 Model Component Metadata

The Model Component metadata is implemented as an ESMF Attribute Package:

- Convention: NUOPC
- Purpose: General
- Includes:
  - CIM Model Component Simulation Description (see for example the Component Attribute packages section in the ESMF v5.2.0rp2 documentation)
- Description: Model component description and nesting metadata.
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Controlled Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verbosity</td>
<td>String value controlling the verbosity of INFO messages.</td>
<td>high, low</td>
</tr>
<tr>
<td>InitializePhaseMap</td>
<td>List of string values, mapping the logical NUOPC initialize phases, of a specific Initialize Phase Definition (IPD) version, to the actual ESMF initialize phase number under which the entry point is registered.</td>
<td>IPDvXXpY=Z, where XX = two-digit revision number, e.g. 01, Y = logical NUOPC phase number, Z = actual ESMF phase number, with Y, Z &gt; 0 and Y, Z &lt; 10</td>
</tr>
<tr>
<td>NestingGeneration</td>
<td>Integer value enumerating nesting level.</td>
<td>0, 1, 2, ...</td>
</tr>
<tr>
<td>Nestling</td>
<td>Integer value enumerating siblings within the same generation.</td>
<td>0, 1, 2, ...</td>
</tr>
<tr>
<td>InitializeDataComplete</td>
<td>String value indicating whether all initialize data dependencies have been satisfied.</td>
<td>false, true</td>
</tr>
<tr>
<td>InitializeDataProgress</td>
<td>String value indicating whether progress is being made resolving initialize data dependencies.</td>
<td>false, true</td>
</tr>
</tbody>
</table>

### 2.3.2 Connector Component Metadata

The Connector Component metadata is implemented as an ESMF Attribute Package:

- Convention: NUOPC
- Purpose: General
- Includes:
  - ESG General (see for example the [Component Attribute packages] section in the ESMF v5.2.0rp2 documentation)
- Description: Basic component description and connection metadata.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Controlled Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verbosity</td>
<td>String value controlling the verbosity of INFO messages.</td>
<td>high, low</td>
</tr>
<tr>
<td>InitializePhaseMap</td>
<td>List of string values, mapping the logical NUOPC initialize phases, of a specific Initialize Phase Definition (IPD) version, to the actual ESMF initialize phase number under which the entry point is registered.</td>
<td>IPDvXXpY=Z, where XX = two-digit revision number, e.g. 01, Y = logical NUOPC phase number, Z = actual ESMF phase number, with Y, Z &gt; 0 and Y, Z &lt; 10</td>
</tr>
<tr>
<td>CplList</td>
<td>List of StandardNames of the connected Fields.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
2.3.3 Field Metadata

The Field metadata is implemented as an ESMF Attribute Package:

- Convention: NUOPC
- Purpose: General
- Includes:
  - ESG General
- Description: Basic Field description with connection and time stamp metadata.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Controlled Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>Connected status.</td>
<td>false, true</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Nine integer values representing ESMF Time object.</td>
<td>N/A</td>
</tr>
<tr>
<td>ProducerConnection</td>
<td>String value indicating connection details.</td>
<td>open, targeted, connected</td>
</tr>
<tr>
<td>ConsumerConnection</td>
<td>String value indicating connection details.</td>
<td>open, targeted, connected</td>
</tr>
<tr>
<td>Updated</td>
<td>String value indicating updated status during initialization.</td>
<td>false, true</td>
</tr>
</tbody>
</table>
2.4 Initialize Phase Definitions

The interaction between NUOPC compliant components during the initialization process is regulated by the Initialize Phase Definition or IPD. The IPDs are versioned, with a higher version number indicating backward compatibility with all previous versions.

There are two perspectives of looking at the IPD. From the driver perspective the IPD regulates the sequence in which it must call the different phases of the Initialize() routines of its child components. To this end the generic NUOPC_Driver component implements support for IPDs up to a version specified in the API documentation.

The other angle of looking at the IPD is from the driver’s child components. From this perspective the IPD assigns specific meaning to each initialize phase. The child components of a driver can be divided into two groups with respect to the meaning the IPD assigns to each initialize phase. In one group are the model, mediator, and driver components, and in the other group are the connector components. The following tables document the meaning of each initialization phase for the two different child component groups for the different IPD versions. The phases are listed in the prescribed sequence used by the driver.

<table>
<thead>
<tr>
<th>IPDv00 label</th>
<th>Child Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPDv00p1</td>
<td>model, mediator, driver</td>
<td>Advertise the import and export Fields.</td>
</tr>
<tr>
<td>IPDv00p1</td>
<td>connector</td>
<td>Construct the CplList Attribute on the connector.</td>
</tr>
<tr>
<td>IPDv00p2</td>
<td>model, mediator, driver</td>
<td>Realize the import and export Fields.</td>
</tr>
<tr>
<td>IPDv00p2</td>
<td>connector</td>
<td>Set the Connected Attribute on each import and export Field. Precompute the RouteHandle.</td>
</tr>
<tr>
<td>IPDv00p3</td>
<td>model, mediator, driver</td>
<td>Check compatibility of the Fields’ Connected status.</td>
</tr>
<tr>
<td>IPDv00p4</td>
<td>model, mediator, driver</td>
<td>Handle Field data initialization. Time stamp the export Fields.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPDv01 label</th>
<th>Child Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPDv01p1</td>
<td>model, mediator, driver</td>
<td>Advertise the import and export Fields.</td>
</tr>
<tr>
<td>IPDv01p1</td>
<td>connector</td>
<td>Construct the CplList Attribute on the connector.</td>
</tr>
<tr>
<td>IPDv01p2</td>
<td>model, mediator, driver</td>
<td>unspecified</td>
</tr>
<tr>
<td>IPDv01p2</td>
<td>connector</td>
<td>Set the Connected Attribute on each import and export Field.</td>
</tr>
<tr>
<td>IPDv01p3</td>
<td>model, mediator, driver</td>
<td>Realize the import and export Fields.</td>
</tr>
<tr>
<td>IPDv01p3</td>
<td>connector</td>
<td>Precompute the RouteHandle.</td>
</tr>
<tr>
<td>IPDv01p4</td>
<td>model, mediator, driver</td>
<td>Check compatibility of the Fields’ Connected status.</td>
</tr>
<tr>
<td>IPDv01p5</td>
<td>model, mediator, driver</td>
<td>Handle Field data initialization. Time stamp the export Fields.</td>
</tr>
<tr>
<td>IPDv02 label</td>
<td>Child Group</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>IPDv02p1</td>
<td>model, mediator, driver</td>
<td>Advertise the import and export Fields.</td>
</tr>
<tr>
<td>IPDv02p1</td>
<td>connector</td>
<td>Construct the CplList Attribute on the connector.</td>
</tr>
<tr>
<td>IPDv02p2</td>
<td>model, mediator, driver</td>
<td>unspecified</td>
</tr>
<tr>
<td>IPDv02p2</td>
<td>connector</td>
<td>Set the Connected Attribute on each import and export Field.</td>
</tr>
<tr>
<td>IPDv02p3</td>
<td>model, mediator, driver</td>
<td>Realize the import and export Fields.</td>
</tr>
<tr>
<td>IPDv02p3</td>
<td>connector</td>
<td>Precompute the RouteHandle.</td>
</tr>
<tr>
<td>IPDv02p4</td>
<td>model, mediator, driver</td>
<td>Check compatibility of the Fields’ Connected status.</td>
</tr>
<tr>
<td>IPDv02p5</td>
<td>model, mediator, driver</td>
<td>Handle Field data initialization. Time stamp the export Fields.</td>
</tr>
</tbody>
</table>

A loop is entered over all those model, mediator, driver Components that use IPDv02 and have unsatisfied data dependencies, repeating the following two steps:

Run ()

- connector
- Loop over all Connectors that connect to the Component that is currently indexed by the outer loop.

IPDv02p5

- model, mediator, driver
- Handle Field data initialization. Time stamp the export Fields.

Repeat these two steps until all data dependencies have been satisfied, or a dead-lock situation is detected.
3 API

3.1 Generic Component: NUOPC_Driver

MODULE:

    module NUOPC_Driver

DESCRIPTION:
Driver component that drives model and connector components. The default is to use explicit time stepping. Each
driver time step, the same sequence of model and connector components’ Run methods are called. The run sequence
is fully customizable.

SUPER:

    ESMF_GridComp

USE DEPENDENCIES:

    use ESMF

SETSERVICES:

    subroutine routine_SetServices(gcomp, rc)
    type(ESMF_GridComp) :: gcomp
    integer, intent(out) :: rc

INITIALIZE:

• phase 0: (REQUIRED, NUOPC PROVIDED)
  – Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition
    (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
    * IPDv00p1 = phase 1: (REQUIRED, NUOPC PROVIDED)

• phase 1: (NUOPC PROVIDED, suitable for: IPDv00p1)
  – Allocate and initialize the internal state.
  – If the internal clock is not yet set, set the default internal clock to be a copy of the incoming clock, if the
    incoming clock is valid.
  – Required specialization to set number of model components, modelCount, in the internal state:
    label_SetModelCount.
  – Allocate internal storage according to modelCount.
  – Optional specialization to provide Model and Connector petList members in the internal state:
    label_SetModelPetList.
  – Create Model components with their import and export States.
  – Attach standard NUOPC Model Component metadata.
  – Create Connector components.
  – Attach standard NUOPC Connector Component metadata.
  – Initialize the default run sequence.
Required specialization to set component services: label_SetModelServices.

- Call into SetServices() for all Model and Connector components.
- Optionally replace the default clock.
- Optionally replace the default run sequence.

- Implement the initialize sequence for the child components, compatible with up to IPD version 01, as documented in section 2.4.

RUN:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - If the incoming clock is valid, set the internal stop time to one time step interval on the incoming clock.
  - Time stepping loop, from current time to stop time, incrementing by time step.
  - For each time step iteration, the Model and Connector components Run() methods are being called according to the run sequence.

FINALIZE:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Optional specialization to finalize driver component: label_Finalize.
  - Execute all Connector components’ Finalize() methods in order.
  - Execute all Model components’ Finalize() methods in order.
  - Destroy all Model components and their import and export states.
  - Destroy all Connector components.
  - Deallocate the run sequence.
  - Deallocate the internal state.

INTERNALSTATE:

label_InternalState

type type_InternalState
  type(type_InternalStateStruct), pointer :: wrap
end type

type InternalState struct
  integer modelCount
  type(type_PetList), pointer modelPetLists(:)
  type(type_PetList), pointer connectorPetLists(:,:)
  type(ESMF_GridComp), pointer modelComp(:)
  type(ESMF_State), pointer modelIS(:), modelES(:)
  type(ESMF_CplComp), pointer connectorComp(:,:)
  type(NUOPC_RunSequence), pointer runSeq(::) size may increase dynamic.
  integer runPhaseToRunSeqMap(10)
  type(ESMF_Clock) driverClock clock of the parent
end type

type type_PetList
  integer, pointer :: petList(:) lists that are set here transfer ownership
end type
3.2 Generic Component: NUOPC_DriverAtmOcn

MODULE:

    module NUOPC_DriverAtmOcn

DESCRIPTION:
This is a specialization of the NUOPC_Driver generic component, driving a coupled Atmosphere-Ocean model. The default is to use explicit time stepping. Each driver time step, the same sequence of Atmosphere, Ocean and connector Run methods are called. The run sequence is fully customizable for cases where explicit time stepping is not suitable.

SUPER:

    NUOPC_Driver

USE DEPENDENCIES:

    use ESMF

SETSERVICES:

    subroutine routine_SetServices(gcomp, rc)
        type(ESMF_GridComp) :: gcomp
        integer, intent(out) :: rc
    end subroutine

INITIALIZE:

    • phase 0: (REQUIRED, NUOPC PROVIDED)
      - Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
        * IPDv00p1 = phase 1: (REQUIRED, NUOPC PROVIDED)
    • phase 1: (NUOPC PROVIDED, suitable for: IPDv00p1)
      - Allocate and initialize the internal state.
      - If the internal clock is not yet set, set the default internal clock to be a copy of the incoming clock, if the incoming clock is valid.
      - Set the number of model components to 2.
      - Allocate internal storage according to modelCount = 2.
      - Optional specialization to provide Model and Connector petList members in the internal state: label_SetModelPetList.
      - Create atm and ocn Model components with their import and export States.
      - Attach standard NUOPC Model Component metadata.
      - Create atm2ocn and ocn2atm Connector components.
      - Attach standard NUOPC Connector Component metadata.
      - Initialize the default run sequence.
      - Required specialization to set component services: label_SetModelServices.
        * Call into SetServices() for the atm, ocn, atm2ocn, and ocn2atm components.
        * Optionally replace the default clock.
* Optionally replace the default run sequence.
  – Implement the initialize sequence for the child components, compatible with up to IPD version 01, as documented in section 2.4

RUN:

  • phase 1: (REQUIRED, NUOPC PROVIDED)
    – If the incoming clock is valid, set the internal stop time to one time step interval on the incoming clock.
    – Time stepping loop, from current time to stop time, incrementing by time step.
    – For each time step iteration, the Run() methods for atm, ocn, atm2ocn, and ocn2atm are being called according to the run sequence.

FINALIZE:

  • phase 1: (REQUIRED, NUOPC PROVIDED)
    – Optional specialization to finalize driver component: label_Finalize.
    – Execute Finalize() for atm2ocn and ocn2atm.
    – Execute Finalize() for atm and ocn.
    – Destroy atm and ocn and their import and export States.
    – Destroy atm2ocn and ocn2atm.
    – Deallocate the run sequence.
    – Deallocate the internal state.

INTERNALSTATE:

label_InternalState
type type_InternalState
  type(type_InternalStateStruct), pointer :: wrap
dend type
type type_InternalStateStruct
  integer, pointer :: atmPetList(:)
  integer, pointer :: ocnPetList(:)
  type(ESMF_GridComp) :: atm
  type(ESMF_GridComp) :: ocn
  type(ESMF_State) :: atmIS, atmES
  type(ESMF_State) :: ocnIS, ocnES
  integer, pointer :: atm2ocnPetList(:)
  integer, pointer :: ocn2atmPetList(:)
  type(ESMF_CplComp) :: atm2ocn, ocn2atm
  type(NUOPC_RunSequence), pointer :: runSeq(:)
dend type
3.3 Generic Component: NUOPC_DriverAtmOcnMed

MODULE:

    module NUOPC_DriverAtmOcnMed

DESCRIPTION:
This is a specialization of the NUOPC_Driver generic component, driving a coupled Atmosphere-Ocean-Mediator model. The default is to use explicit time stepping. Each driver time step, the same sequence of Atmosphere, Ocean, Mediator, and the connector Run methods are called. The run sequence is fully customizable for cases where explicit time stepping is not suitable.

SUPER:
    NUOPC_Driver

USE DEPENDENCIES:
    use ESMF

SETSERVICES:

    subroutine routine_SetServices(gcomp, rc)
        type(ESMF_GridComp) :: gcomp
        integer, intent(out) :: rc
    end subroutine

INITIALIZE:

• phase 0: (REQUIRED, NUOPC PROVIDED)
  - Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
    * IPDv00p1 = phase 1: (REQUIRED, NUOPC PROVIDED)

• phase 1: (NUOPC PROVIDED, suitable for: IPDv00p1)
  - Allocate and initialize the internal state.
  - If the internal clock is not yet set, set the default internal clock to be a copy of the incoming clock, if the incoming clock is valid.
  - Set the number of model components to 3.
  - Allocate internal storage according to modelCount = 3.
  - Optional specialization to provide Model and Connector petList members in the internal state: label_SetModelPetList.
  - Create atm, ocn, and med components with their import and export States.
  - Attach standard NUOPC Model Component metadata.
  - Create atm2ocn, atm2med, ocn2atm, ocn2atm, med2atm, and med2ocn Connector components.
  - Attach standard NUOPC Connector Component metadata.
  - Initialize the default run sequence.
  - Required specialization to set component services: label_SetModelServices.
    * Call into SetServices() for the atm, ocn, med, atm2ocn, atm2med, ocn2atm, ocn2atm, med2atm, and med2ocn components.
* Optionally replace the default clock.
* Optionally replace the default run sequence.

- Implement the initialize sequence for the child components, compatible with up to IPD version 01, as documented in section 2.4.

RUN:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - If the incoming clock is valid, set the internal stop time to one time step interval on the incoming clock.
  - Time stepping loop, from current time to stop time, incrementing by time step.
  - For each time step iteration, the Run() methods for atm, ocn, med, atm2ocn, atm2med, ocn2atm, ocn2atm, med2atm, and med2ocn are being called according to the run sequence.

FINALIZE:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Optional specialization to finalize driver component: label_Finalize.
  - Execute Finalize() for atm2ocn, atm2med, ocn2atm, ocn2atm, med2atm, and med2ocn.
  - Execute Finalize() for atm, ocn, and med.
  - Destroy atm, ocn, and med and their import and export States.
  - Destroy atm2ocn, atm2med, ocn2atm, ocn2atm, med2atm, and med2ocn.
  - Deallocate the run sequence.
  - Deallocation the internal state.

INTERNALSTATE:

label_InternalState

type type_InternalState
  type(type_InternalStateStruct), pointer :: wrap
end type

type type_InternalStateStruct
  integer, pointer :: atmPetList(:)
  integer, pointer :: ocnPetList(:)
  integer, pointer :: medPetList(:)
  type(ESMF_GridComp) :: atm
  type(ESMF_GridComp) :: ocn
  type(ESMF_GridComp) :: med
  type(ESMF_State) :: atmIS, atmES
  type(ESMF_State) :: ocnIS, ocnES
  type(ESMF_State) :: medIS, medES
  integer, pointer :: atm2medPetList(:)
  integer, pointer :: ocn2medPetList(:)
  integer, pointer :: med2atmPetList(:)
  integer, pointer :: med2ocnPetList(:)
  type(ESMF_CplComp) :: atm2med, ocn2med
  type(ESMF_CplComp) :: med2atm, med2ocn
  type(NUOPC_RunSequence), pointer :: runSeq(:)
end type

16
3.4 Generic Component: NUOPC_ModelBase

MODULE:

```fortran
module NUOPC_ModelBase
```

DESCRIPTION:
Model component with a default explicit time dependency. Each time the Run method is called the model integrates one timeStep forward on the provided Clock. The Clock must be advanced between Run calls. The component’s Run method flags incompatibility if the current time of the incoming Clock does not match the current time of the model.

SUPER:

```fortran
ESMF_GridComp
```

USE DEPENDENCIES:

```fortran
use ESMF
```

SETSERVICES:

```fortran
subroutine routine_SetServices(gcomp, rc)
  type(ESMF_GridComp) :: gcomp
  integer, intent(out) :: rc
```

INITIALIZE:

- phase 0: (REQUIRED, NUOPC PROVIDED)
  - Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
    * IPDv00p1 = phase 1: (REQUIRED, IMPLEMENTOR PROVIDED)
    * IPDv00p2 = phase 2: (REQUIRED, IMPLEMENTOR PROVIDED)
    * IPDv00p3 = phase 3: (REQUIRED, IMPLEMENTOR PROVIDED)
    * IPDv00p4 = phase 4: (REQUIRED, IMPLEMENTOR PROVIDED)

RUN:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Allocate internal state memory.
  - Assign the driverClock member in the internal state as an alias to the incoming clock.
  - Optional specialization to check and set the internal clock against the incoming clock: label_SetRunClock.
  - Alternatively use the default specialization: check that internal clock and incoming clock agree on current time and that the time step of the incoming clock is a multiple of the internal clock time step. Under these conditions set the internal stop time to one time step interval on the incoming clock. Otherwise exit with error, flagging incompatibility.
  - Optional specialization to check Fields in import State: label_CheckImport.
  - Alternatively use the default specialization: check that all import Fields are at the current time of the internal clock.
Model time stepping loop, starting at current time, running to stop time on the internal clock using the internal Clock time step. Timestamp the Fields in the export State at the beginning of each iteration.

- Required specialization to advance the model each time step: `label_Advance`.
- Optional specialization to timestamp the Fields in the export State: `label_TimestampExport`.
- Deallocate internal state memory.

FINALIZE:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Optionally overwrite the provided NOOP with model finalization code.

INTERNALSTATE:

```plaintext
label_InternalState
type type_InternalState
  type(type_InternalStateStruct), pointer :: wrap
type type_InternalStateStruct
type(ESMF_Clock) :: driverClock
end type
```

3.5 Generic Component: NUOPC_Model

MODULE:

```plaintext
module NUOPC_Model
end module
```

DESCRIPTION:
Model component with a default explicit time dependency. Each time the `Run` method is called the model integrates one timeStep forward on the provided Clock. The Clock must be advanced between `Run` calls. The component’s `Run` method flags incompatibility if the current time of the incoming Clock does not match the current time of the model.

SUPER:

`NUOPC_ModelBase`

USE DEPENDENCIES:

```plaintext
use ESMF
```

SETSERVICES:

```plaintext
subroutine routine_SetServices(gcomp, rc)
type(ESMF_GridComp) :: gcomp
integer, intent(out) :: rc
```

INITIALIZE:
• phase 0: (REQUIRED, NUOPC PROVIDED)
  – Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
    * IPDv00p1 = phase 1: (REQUIRED, IMPLEMENTOR PROVIDED)
    * IPDv00p2 = phase 2: (REQUIRED, IMPLEMENTOR PROVIDED)
    * IPDv00p3 = phase 3: (REQUIRED, NUOPC PROVIDED)
    * IPDv00p4 = phase 4: (REQUIRED, NUOPC PROVIDED)

• phase 3: (NUOPC PROVIDED, suitable for: IPDv00p3, IPDv01p4, IPDv02p4)
  – If the model internal clock is found to be not set, then set the model internal clock as a copy of the incoming clock.
  – Optional specialization to set the internal clock and/or alarms: label_SetClock.
  – Check compatibility, ensuring all advertised import Fields are connected.

• phase 4: (NUOPC PROVIDED, suitable for: IPDv00p4, IPDv01p5)
  – Optional specialization to initialize export Fields: label_DataInitialize
  – Time stamp Fields in export State for compatibility checking.

• phase 5: (NUOPC PROVIDED, suitable for: IPDv02p5)
  – Optional specialization to initialize export Fields: label_DataInitialize
  – Time stamp Fields in export State for compatibility checking.
  – Set Component metadata used to resolve initialize data dependencies.

RUN:
• phase 1: (REQUIRED, NUOPC PROVIDED)
  – Allocate internal state memory.
  – Assign the driverClock member in the internal state as an alias to the incoming clock.
  – Optional specialization to check and set the internal clock against the incoming clock: label_SetRunClock.
  – Alternatively use the default specialization: check that internal clock and incoming clock agree on current time and that the time step of the incoming clock is a multiple of the internal clock time step. Under these conditions set the internal stop time to one time step interval on the incoming clock. Otherwise exit with error, flagging incompatibility.
  – Optional specialization to check Fields in import State: label_CheckImport.
  – Alternatively use the default specialization: check that all import Fields are at the current time of the internal clock.
  – Model time stepping loop, starting at current time, running to stop time on the internal clock using the internal Clock time step.
  – Required specialization to advance the model each time step: label_Advance.
  – Timestamp all export Fields at the current time of the internal clock.
  – Deallocate internal state memory.

FINALIZE:
• phase 1: (REQUIRED, NUOPC PROVIDED)
– Optionally overwrite the provided NOOP with model finalization code.

INTERNALSTATE:

label_InternalState

type type_InternalState
  type(type_InternalStateStruct), pointer :: wrap
end type

type type_InternalStateStruct
  type(ESMF_Clock) :: driverClock
end type

3.6 Generic Component: NUOPC_Mediator

MODULE:

module NUOPC_Mediator

DESCRIPTION:
Mediator component with a default *explicit* time dependency. Each time the Run method is called, the time stamp on the imported Fields must match the current time (on both the incoming and internal Clock). Before returning, the Mediator time stamps the exported Fields with the same current time, before advancing the internal Clock one timeStep forward.

SUPER:

  NUOPC_ModelBase

USE DEPENDENCIES:

  use ESMF

SETSERVICES:

  subroutine routine_SetServices(gcomp, rc)
    type(ESMF_GridComp) :: gcomp
    integer, intent(out) :: rc
  end subroutine

INITIALIZE:

  • phase 0: (REQUIRED, NUOPC PROVIDED)
    – Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 00 (see section 2.4 for a precise definition), with the following mapping:
      * IPDv00p1 = phase 1: (REQUIRED, IMPLEMENTOR PROVIDED)
        - Advertise Fields in import and export States.
      * IPDv00p2 = phase 2: (REQUIRED, IMPLEMENTOR PROVIDED)
        - Realize advertised Fields in import and export States.
      * IPDv00p3 = phase 3: (REQUIRED, NUOPC PROVIDED)
Additional initialization phase.

* IPDv00p4 = phase 4: (REQUIRED, NUOPC PROVIDED)
  - Additional initialization phase.

- phase 3: (REQUIRED, NUOPC PROVIDED)
  - Set the Mediator internal clock as a copy of the incoming clock.
  - Check compatibility, ensuring all advertised import Fields are connected.

- phase 4: (REQUIRED, NUOPC PROVIDED)
  - Optional specialization to initialize export Fields: label_DataInitialize
  - Time stamp Fields in import and export States for compatibility checking.

RUN:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Allocate internal state memory.
  - Assign the driverClock member in the internal state as an alias to the incoming clock.
  - Check that internal clock and incoming clock agree on current time and that the time step of the incoming clock is a multiple of the internal clock time step (remember that the generic InitializePhase2 set the Mediator internal clock identical to the incoming clock). Under these conditions, set the internal stop time to one time step interval on the incoming clock. Otherwise exit with error, flagging incompatibility.
  - Optional specialization to check Fields in import State: label_CheckImport.
  - Alternatively use the default specialization: check that all import Fields are at the current time of the internal clock.
  - Timestamp all export Fields at the current time of the internal clock, i.e. the current time of the incoming clock.
  - Mediator time step forward on the internal Clock, which is the same time step as on the incoming Clock. This prepares the internal clock for the next iteration.
  - Required specialization to mediate the Fields: label_Advance.
  - Deallocate internal state memory.

FINALIZE:

- phase 1: (REQUIRED, NUOPC PROVIDED)
  - Optionally overwrite the provided NOOP with Mediator finalization code.

INTERNALSTATE:

```fortran
label_InternalState
type type_InternalState
    type(type_InternalStateStruct), pointer :: wrap
end type

type type_InternalStateStruct
    type(ESMF_Clock) :: driverClock
end type
```

21
3.7 Generic Component: NUOPC_Connector

MODULE:

module NUOPC_Connector

DESCRIPTION:
Connector component that uses a default bilinear regrid method during Run to transfer data from the connected import Fields to the connected export Fields.

SUPER:

ESMF_CplComp

USE DEPENDENCIES:

use ESMF

SETSERVICES:

subroutine routine_SetServices(cplcomp, rc)
  type(ESMF_CplComp) :: cplcomp
  integer, intent(out) :: rc

INITIALIZE:

• phase 0: (REQUIRED, NUOPC PROVIDED)
  
  - Initialize the InitializePhaseMap Attribute according to the NUOPC Initialize Phase Definition (IPD) version 01 (see section 2.4 for a precise definition), with the following mapping:
    * IPDv01p1 = phase 1: (REQUIRED, NUOPC PROVIDED)
    * IPDv01p2 = phase 2: (REQUIRED, NUOPC PROVIDED)
    * IPDv01p3 = phase 3: (REQUIRED, NUOPC PROVIDED)

• phase 1: (NUOPC PROVIDED, suitable for: IPDv01p1, IPDv02p1)
  
  - Construct a list of matching Field pairs between import and export State based on the StandardName Field metadata.
  - Store this list of StandardName entries in the CplList attribute of the Connector Component metadata.

• phase 2: (NUOPC PROVIDED, suitable for: IPDv01p2, IPDv02p2)
  
  - Allocate and initialize the internal state.
  - Use the CplList attribute to construct srcFields and dstFields FieldBundles in the internal state that hold matched Field pairs.
  - Set the Connected attribute to true in the Field metadata for each Field that is added to the srcFields and dstFields FieldBundles.

• phase 3: (NUOPC PROVIDED, suitable for: IPDv01p3, IPDv02p3)
  
  - Use the CplList attribute to construct srcFields and dstFields FieldBundles in the internal state that hold matched Field pairs.
- Set the Connected attribute to true in the Field metadata for each Field that is added to the srcFields and dstFields FieldBundles.

- Optional specialization to precompute a Connector operation: label_ComputeRouteHandle. Simple custom implementations store the precomputed communication RouteHandle in the rh member of the internal state. More complex implementations use the state member in the internal state to store auxiliary Fields, FieldBundles, and RouteHandles.

- By default (if label_ComputeRouteHandle was not provided) precompute the Connector RouteHandle as a bilinear Regrid operation between srcFields and dstFields, with unmappedaction set to ESMF_UNMAPPEDACTION_IGNORE. The resulting RouteHandle is stored in the rh member of the internal state.

RUN:

- phase 1: (REQUIRED, NUOPC PROVIDED)

  - Optional specialization to execute a Connector operation: label_ExecuteRouteHandle. Simple custom implementations access the srcFields, dstFields, and rh members of the internal state to implement the required data transfers. More complex implementations access the state member in the internal state, which holds the auxiliary Fields, FieldBundles, and RouteHandles that potentially were added during the optional label_ComputeRouteHandle method during initialize.

  - By default (if label_ExecuteRouteHandle was not provided) execute the precomputed Connector RouteHandle between srcFields and dstFields.

  - Update the time stamp on the Fields in dstFields to match the time stamp on the Fields in srcFields.

FINALIZE:

- phase 1: (REQUIRED, NUOPC PROVIDED)

  - Optional specialization to release a Connector operation: label_ReleaseRouteHandle.

  - By default (if label_ReleaseRouteHandle was not provided) release the precomputed Connector RouteHandle.

  - Destroy the internal state members.

  - Deallocate the internal state.

INTERNALSTATE:

    label_InternalState

    type type_InternalState
    type(type_InternalStateStruct), pointer :: wrap
end type

    type type_InternalStateStruct
    type(ESMF_FieldBundle) :: srcFields
    type(ESMF_FieldBundle) :: dstFields
    type(ESMF_RouteHandle) :: rh
    type(ESMF_State) :: state
end type
3.8 Utility Class: NUOPC_RunSequence

The NUOPC_RunSequence class provides a unified data structure that allows simple as well as complex time loops to be encoded and executed. There are entry points that allow different run phases to be mapped against distinctly different time loops.

Figure 2 depicts the data structures surrounding the NUOPC_RunSequence, starting with the InternalState of the NUOPC_Driver generic component.

3.8.1 NUOPC_RunElementAdd - Add a RunElement to the end of a RunSequence

INTERFACE:

    subroutine NUOPC_RunElementAdd(runSeq, i, j, phase, rc)

ARGUMENTS:

    type(NUOPC_RunSequence), intent(inout), target :: runSeq
    integer,    intent(in)                   :: i, j, phase
    integer,    optional, intent(out)       :: rc
3.8.2 NUOPC_RunElementPrint - Print info about a RunElement object

INTERFACE:

    subroutine NUOPC_RunElementPrint(runElement, rc)

ARGUMENTS:

    type(NUOPC_RunElement), intent(in) :: runElement
    integer, optional, intent(out) :: rc

DESCRIPTION:

Write information about runElement into the default log file.

3.8.3 NUOPC_RunSequenceAdd - Add more RunSequences to a RunSequence vector

INTERFACE:

    subroutine NUOPC_RunSequenceAdd(runSeq, addCount, rc)

ARGUMENTS:

    type(NUOPC_RunSequence), pointer :: runSeq(:)
    integer, intent(in) :: addCount
    integer, optional, intent(out) :: rc

DESCRIPTION:

The incoming RunSequence vector runSeq is extended by addCount more RunSequence objects. The existing RunSequence objects are copied to the front of the new vector before the old vector is deallocated.

3.8.4 NUOPC_RunSequenceDeallocate - Deallocate an entire RunSequence vector

INTERFACE:
! Private name; call using NUOPC_RunSequenceDeallocate()
subroutine NUOPC_RunSequenceArrayDeall(runSeq, rc)

ARGUMENTS:

    type(NUOPC_RunSequence), pointer           :: runSeq(:)
    integer, optional,           intent(out) :: rc

DESCRIPTION:

Dealocate all of the RunElements in all of the RunSequence defined in the runSeq vector.

3.8.5 NUOPC_RunSequenceDeallocate - Deallocate a single RunSequence object

INTERFACE:

! Private name; call using NUOPC_RunSequenceDeallocate()
subroutine NUOPC_RunSequenceSingleDeall(runSeq, rc)

ARGUMENTS:

    type(NUOPC_RunSequence), intent(inout) :: runSeq
    integer, optional,           intent(out) :: rc

DESCRIPTION:

Dealocate all of the RunElements in the RunSequence defined by runSeq.

3.8.6 NUOPC_RunSequenceIterate - Iterate through a RunSequence

INTERFACE:

    function NUOPC_RunSequenceIterate(runSeq, runSeqIndex, runElement, rc)

RETURN VALUE:

    logical :: NUOPC_RunSequenceIterate

ARGUMENTS:

    type(NUOPC_RunSequence), pointer           :: runSeq(:)
    integer,           intent(in) :: runSeqIndex
    type(NUOPC_RunElement), pointer           :: runElement
    integer, optional,           intent(out) :: rc
Iterate through the RunSequence that is in position runSeqIndex in the runSeq vector. If runElement comes in unassociated, the iteration starts from the beginning. Otherwise this call takes one forward step relative to the incoming runElement, returning the next RunElement in runElement. In either case, the logical function return value is .true. if the end of iteration has not been reached by the forward step, and .false. if the end of iteration has been reached. The returned runElement is only valid for a function return value of .true..

3.8.7 NUOPC_RunSequencePrint - Print info about a single RunSequence object

INTERFACE:

! Private name; call using NUOPC_RunSequencePrint()
subroutine NUOPC_RunSequenceSinglePrint(runSeq, rc)

ARGUMENTS:

   type(NUOPC_RunSequence), intent(in) :: runSeq
   integer, optional,         intent(out) :: rc

DESCRIPTION:

Write information about runSeq into the default log file.

3.8.8 NUOPC_RunSequencePrint - Print info about a RunSequence vector

INTERFACE:

! Private name; call using NUOPC_RunSequencePrint()
subroutine NUOPC_RunSequenceArrayPrint(runSeq, rc)

ARGUMENTS:

   type(NUOPC_RunSequence), pointer :: runSeq(:)
   integer, optional,         intent(out) :: rc

DESCRIPTION:

Write information about the whole runSeq vector into the default log file.
3.8.9 NUOPC_RunSequenceSet - Set values inside a RunSequence object

INTERFACE:

subroutine NUOPC_RunSequenceSet(runSeq, clock, rc)

ARGUMENTS:

type(NUOPC_RunSequence), intent(inout) :: runSeq
type(ESMF_Clock), intent(in) :: clock
integer, optional, intent(out) :: rc

DESCRIPTION:

Set the Clock member in runSeq.
3.9 Utility Routines

3.9.1 NUOPC_ClockCheckSetClock - Check a Clock for compatibility

INTERFACE:

```fortran
subroutine NUOPC_ClockCheckSetClock(setClock, checkClock, &
   setStartTimeToCurrent, rc)
```

ARGUMENTS:

```fortran
  type(ESMF_Clock), intent(inout) :: setClock
  type(ESMF_Clock), intent(in) :: checkClock
  logical, intent(in), optional :: setStartTimeToCurrent
  integer, intent(out), optional :: rc
```

DESCRIPTION:

Compares setClock to checkClock to make sure they match in their current Time. Further ensures that checkClock's timeStep is a multiple of setClock's timeStep. If both these condition are satisfied then the stop-Time of the setClock is set one checkClock's timeStep ahead of the current Time, taking into account the direction of the Clock.

By default the startTime of the setClock is not modified. However, if setStartTimeToCurrent == .true. the startTime of setClock is set to the currentTime of checkClock.

3.9.2 NUOPC_ClockInitialize - Initialize a new Clock from Clock and stabilityTimeStep

INTERFACE:

```fortran
function NUOPC_ClockInitialize(externalClock, stabilityTimeStep, rc)
```

RETURN VALUE:

```fortran
  type(ESMF_Clock) :: NUOPC_ClockInitialize
```

ARGUMENTS:

```fortran
  type(ESMF_Clock) :: externalClock
  type(ESMF_TimeInterval), intent(in), optional :: stabilityTimeStep
  integer, intent(out), optional :: rc
```

DESCRIPTION:

Returns a new Clock instance that is a copy of the incoming Clock, but potentially with a smaller timestep. The timestep is chosen so that the timestep of the incoming Clock (externalClock) is a multiple of the new Clock’s timestep, and at the same time the new timestep is <= the stabilityTimeStep.
3.9.3 NUOPC_ClockPrintCurrTime - Formatted print of current time

INTERFACE:

    subroutine NUOPC_ClockPrintCurrTime(clock, string, unit, rc)

ARGUMENTS:

    type(ESMF_Clock), intent(in) :: clock
    character(*), intent(in), optional :: string
    character(*), intent(out), optional :: unit
    integer, intent(out), optional :: rc

DESCRIPTION:

Writes the formatted current time of clock to unit. Prepends string if provided. If unit is present it must be an internal unit, i.e. a string variable. If unit is not present then the output is written to the default external unit (typically that would be stdout).

3.9.4 NUOPC_ClockPrintStartTime - Formatted print of start time

INTERFACE:

    subroutine NUOPC_ClockPrintStartTime(clock, string, unit, rc)

ARGUMENTS:

    type(ESMF_Clock), intent(in) :: clock
    character(*), intent(in), optional :: string
    character(*), intent(out), optional :: unit
    integer, intent(out), optional :: rc

DESCRIPTION:

Writes the formatted start time of clock to unit. Prepends string if provided. If unit is present it must be an internal unit, i.e. a string variable. If unit is not present then the output is written to the default external unit (typically that would be stdout).

3.9.5 NUOPC_ClockPrintStopTime - Formatted print of stop time

INTERFACE:

    subroutine NUOPC_ClockPrintStopTime(clock, string, unit, rc)
ARGUMENTS:

```
type(ESMF_Clock), intent(in) :: clock
character(*),   intent(in), optional :: string
character(*),   intent(out), optional :: unit
integer,        intent(out), optional :: rc
```

DESCRIPTION:

Writes the formatted stop time of `clock` to `unit`. Prepends `string` if provided. If `unit` is present it must be an internal unit, i.e. a string variable. If `unit` is not present then the output is written to the default external unit (typically that would be stdout).

---

3.9.6  NUOPC_CplCompAreServicesSet - Check if SetServices was called

INTERFACE:

```
function NUOPC_CplCompAreServicesSet(comp, rc)
```

RETURN VALUE:

```
logical :: NUOPC_CplCompAreServicesSet
```

ARGUMENTS:

```
type(ESMF_CplComp), intent(in) :: comp
integer,        intent(out), optional :: rc
```

DESCRIPTION:

Returns `.true.` if SetServices has been called for `comp`. Otherwise returns `.false.`.

---

3.9.7  NUOPC_CplCompAttributeAdd - Add the NUOPC CplComp Attributes

INTERFACE:

```
subroutine NUOPC_CplCompAttributeAdd(comp, rc)
```

ARGUMENTS:

```
type(ESMF_CplComp), intent(inout) :: comp
integer,        intent(out), optional :: rc
```
DESCRIPTION:

Adds standard NUOPC Attributes to a Coupler Component. Checks the provided importState and exportState arguments for matching Fields and adds the list as "CplList" Attribute.

This adds the standard NUOPC Coupler Attribute package: convention="NUOPC", purpose="General" to the Field. The NUOPC Coupler Attribute package extends the ESG Component Attribute package: convention="ESG", purpose="General".

The arguments are:

- **comp** The ESMF_CplComp object to which the Attributes are added.
- **[rc]** Return code; equals ESMF_SUCCESS if there are no errors.

### 3.9.8 NUOPC_CplCompAttributeGet - Get a NUOPC CplComp Attribute

**INTERFACE:**

```fortran
subroutine NUOPC_CplCompAttributeGet(comp, cplList, cplListSize, rc)
```

**ARGUMENTS:**

- `type(ESMF_CplComp), intent(in) :: comp`
- `character(*), intent(out), optional :: cplList(:)`
- `integer, intent(out), optional :: cplListSize`
- `integer, intent(out), optional :: rc`

**DESCRIPTION:**

Accesses the "CplList" Attribute inside of **comp** using the convention NUOPC and purpose General. Returns with error if the Attribute is not present or not set.

### 3.9.9 NUOPC_CplCompAttributeSet - Set the NUOPC CplComp Attributes

**INTERFACE:**

```fortran
subroutine NUOPC_CplCompAttributeSet(comp, importState, exportState, rc)
```

**ARGUMENTS:**

- `type(ESMF_CplComp), intent(inout) :: comp`
- `type(ESMF_State), intent(in) :: importState`
- `type(ESMF_State), intent(in) :: exportState`
- `integer, intent(out), optional :: rc`
DESCRIPTION:

Checks the provided importState and exportState arguments for matching Fields and sets the coupling list as "CplList" Attribute in comp.

The arguments are:

**comp** The ESMF_CplComp object to which the Attributes are set.

**importState** Import State.

**exportState** Export State.

**[rc]** Return code; equals ESMF_SUCCESS if there are no errors.

3.9.10 NUOPC_FieldAttributeAdd - Add the NUOPC Field Attributes

INTERFACE:

```fortran
subroutine NUOPC_FieldAttributeAdd(field, StandardName, Units, LongName, &
    ShortName, Connected, rc)
```

ARGUMENTS:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type(ESMF_Field)</td>
<td>field</td>
</tr>
<tr>
<td>character(*) , intent(in)</td>
<td>StandardName</td>
</tr>
<tr>
<td>character(*) , intent(in), optional</td>
<td>Units</td>
</tr>
<tr>
<td>character(*) , intent(in), optional</td>
<td>LongName</td>
</tr>
<tr>
<td>character(*) , intent(in), optional</td>
<td>ShortName</td>
</tr>
<tr>
<td>character(*) , intent(in), optional</td>
<td>Connected</td>
</tr>
<tr>
<td>integer, intent(out), optional</td>
<td>rc</td>
</tr>
</tbody>
</table>

DESCRIPTION:

Adds standard NUOPC Attributes to a Field object. Checks the provided arguments against the NUOPC Field Dictionary. Omitted optional information is filled in using defaults out of the NUOPC Field Dictionary.

This adds the standard NUOPC Field Attribute package: convention="NUOPC", purpose="General" to the Field. The NUOPC Field Attribute package extends the ESG Field Attribute package: convention="ESG", purpose="General".

The arguments are:

**field** The ESMF_Field object to which the Attributes are added.

**StandardName** The StandardName of the Field. Must be a StandardName found in the NUOPC Field Dictionary.

**[Units]** The Units of the Field. Must be convertible to the canonical units specified in the NUOPC Field Dictionary for the specified StandardName. If omitted, the default is to use the canonical units associated with the StandardName in the NUOPC Field Dictionary.

**[LongName]** The LongName of the Field. NUOPC does not restrict the value of this variable. If omitted, the default is to use the LongName associated with the StandardName in the NUOPC Field Dictionary.
[ShortName] The ShortName of the Field. NUOPC does not restrict the value of this variable. If omitted, the default is to use the ShortName associated with the StandardName in the NUOPC Field Dictionary.

[Connected] The connection status of the Field. Must be one of the NUOPC supported values: false or true. If omitted, the default is a connected status of false.

[rc] Return code; equals ESMF_SUCCESS if there are no errors.

3.9.11 NUOPC_FieldAttributeGet - Get a NUOPC Field Attribute

INTERFACE:

    subroutine NUOPC_FieldAttributeGet(field, name, value, rc)

ARGUMENTS:

    type(ESMF_Field), intent(in) :: field
    character(*), intent(in)    :: name
    character(*), intent(out)   :: value
    integer, intent(out), optional :: rc

DESCRIPTION:

    Accesses the Attribute name inside of field using the convention NUOPC and purpose General. Returns with error if the Attribute is not present or not set.

3.9.12 NUOPC_FieldAttributeSet - Set a NUOPC Field Attribute

INTERFACE:

    subroutine NUOPC_FieldAttributeSet(field, name, value, rc)

ARGUMENTS:

    type(ESMF_Field)            :: field
    character(*), intent(in)    :: name
    character(*), intent(in)    :: value
    integer, intent(out), optional :: rc

DESCRIPTION:

    Set the Attribute name inside of field using the convention NUOPC and purpose General.
3.9.13 NUOPC_FieldBundleUpdateTime - Update the time stamp on all Fields in a FieldBundle

INTERFACE:

subroutine NUOPC_FieldBundleUpdateTime(srcFields, dstFields, rc)

ARGUMENTS:

type(ESMF_FieldBundle), intent(in) :: srcFields
type(ESMF_FieldBundle), intent(inout) :: dstFields
integer, intent(out), optional :: rc

DESCRIPTION:

Updates the time stamp on all Fields in the dstFields FieldBundle to be the same as in the dstFields FieldBundle.

3.9.14 NUOPC_FieldDictionaryAddEntry - Add an entry to the NUOPC Field dictionary

INTERFACE:

subroutine NUOPC_FieldDictionaryAddEntry(standardName, canonicalUnits, 
defaultLongName, defaultShortName, rc)

ARGUMENTS:

character(*), intent(in) :: standardName
character(*), intent(in) :: canonicalUnits
character(*), intent(in), optional :: defaultLongName
character(*), intent(in), optional :: defaultShortName
integer, intent(out), optional :: rc

DESCRIPTION:

Adds an entry to the NUOPC Field dictionary. If necessary the dictionary is first set up.

3.9.15 NUOPC_FieldDictionaryGetEntry - Get information about a NUOPC Field dictionary entry

INTERFACE:

subroutine NUOPC_FieldDictionaryGetEntry(standardName, canonicalUnits, 
defaultLongName, defaultShortName, rc)
ARGUMENTS:

character( * ), intent(in) :: standardName
character( * ), intent(out), optional :: canonicalUnits
character( * ), intent(out), optional :: defaultLongName
character( * ), intent(out), optional :: defaultShortName
integer, intent(out), optional :: rc

DESCRIPTION:

Returns the canonical units, the default LongName and the default ShortName that the NUOPC Field dictionary associates with a StandardName.

3.9.16 NUOPC_FieldDictionaryHasEntry - Check whether the NUOPC Field dictionary has a specific entry

INTERFACE:

function NUOPC_FieldDictionaryHasEntry(standardName, rc)

RETURN VALUE:

logical :: NUOPC_FieldDictionaryHasEntry

ARGUMENTS:

character( * ), intent(in) :: standardName
integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if the NUOPC Field dictionary has an entry with the specified StandardName, .false. otherwise.

3.9.17 NUOPC_FieldDictionarySetup - Setup the NUOPC Field dictionary

INTERFACE:

subroutine NUOPC_FieldDictionarySetup(rc)

ARGUMENTS:

integer, intent(out), optional :: rc

DESCRIPTION:

Setup the NUOPC Field dictionary.
3.9.18 NUOPC_FieldIsAtTime - Check if the Field is at the given Time

INTERFACE:

    function NUOPC_FieldIsAtTime(field, time, rc)

RETURN VALUE:

    logical :: NUOPC_FieldIsAtTime

ARGUMENTS:

    type(ESMF_Field), intent(in) :: field
    type(ESMF_Time), intent(in) :: time
    integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if the Field has a timestamp that matches time. Otherwise returns .false.

3.9.19 NUOPC_FillCplList - Fill the cplList according to matching Fields

INTERFACE:

    subroutine NUOPC_FillCplList(importState, exportState, cplList, rc)

ARGUMENTS:

    type(ESMF_State), intent(in) :: importState
    type(ESMF_State), intent(in) :: exportState
    character(ESMF_MAXSTR), pointer :: cplList(:)
    integer, intent(out), optional :: rc

DESCRIPTION:

Constructs a list of matching StandardNames of Fields in the importState and exportState. Returns this list in cplList.

The pointer argument cplList must enter this method unassociated. On return, the deallocation of the potentially associated pointer becomes the user responsibility.

3.9.20 NUOPC_GridCompAreServicesSet - Check if SetServices was called

INTERFACE:
function NUOPC_GridCompAreServicesSet(comp, rc)

RETURN VALUE:

logical :: NUOPC_GridCompAreServicesSet

ARGUMENTS:

type(ESMF_GridComp), intent(in) :: comp
integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if SetServices has been called for comp. Otherwise returns .false..

3.9.21 NUOPC_GridCompAttributeAdd - Add the NUOPC GridComp Attributes

INTERFACE:

subroutine NUOPC_GridCompAttributeAdd(comp, rc)

ARGUMENTS:

type(ESMF_GridComp) :: comp
integer, intent(out), optional :: rc

DESCRIPTION:

Adds standard NUOPC Attributes to a Gridded Component.

This adds the standard NUOPC GridComp Attribute package: convention="NUOPC", purpose="General" to the Gridded Component. The NUOPC GridComp Attribute package extends the CIM Component Attribute package: convention="CIM 1.5", purpose="ModelComp".

3.9.22 NUOPC_GridCompCheckSetClock - Check Clock compatibility and set stopTime

INTERFACE:

subroutine NUOPC_GridCompCheckSetClock(comp, externalClock, rc)

ARGUMENTS:

type(ESMF_GridComp), intent(inout) :: comp
type(ESMF_Clock), intent(in) :: externalClock
integer, intent(out), optional :: rc
DESCRIPTION:

Compares externalClock to the Component internal Clock to make sure they match in their current Time. Further ensures that the external Clock’s timeStep is a multiple of the internal Clock’s timeStep. If both these condition are satisfied then the stopTime of the internal Clock is set to be reachable in one timeStep of the external Clock, taking into account the direction of the Clock.

3.9.23 NUOPC_GridCompSetClock - Initialize and set the internal Clock of a GridComp

INTERFACE:

```fortran
subroutine NUOPC_GridCompSetClock(comp, externalClock, stabilityTimeStep, &
rc)
ARGUMENTS:

  type(ESMF_GridComp), intent(inout) :: comp
  type(ESMF_Clock), intent(in) :: externalClock
  type(ESMF_TimeInterval), intent(in), optional :: stabilityTimeStep
  integer, intent(out), optional :: rc
```

DESCRIPTION:

Sets the Component internal Clock as a copy of externalClock, but with a timeStep that is less than or equal to the stabilityTimeStep. At the same time ensures that the timeStep of the external Clock is a multiple of the internal Clock’s timeStep. If the stabilityTimeStep argument is not provided then the internal Clock will simply be set as a copy of the externalClock.

3.9.24 NUOPC_GridCreateSimpleXY - Create a simple XY cartesian Grid

INTERFACE:

```fortran
function NUOPC_GridCreateSimpleXY(x_min, y_min, x_max, y_max, &
i_count, j_count, rc)
RETURN VALUE:

  type(ESMF_Grid):: NUOPC_GridCreateSimpleXY
ARGUMENTS:

  real(ESMF_KIND_R8), intent(in) :: x_min, x_max, y_min, y_max
  integer, intent(in) :: i_count, j_count
  integer, intent(out), optional :: rc
```
DESCRIPTION:

Creates and returns a very simple XY cartesian Grid.

3.9.25 NUOPC_IsCreated - Check whether an ESMF object has been created

INTERFACE:

! call using generic interface: NUOPC_IsCreated
function NUOPC_ClockIsCreated(clock, rc)

RETURN VALUE:

logical :: NUOPC_ClockIsCreated

ARGUMENTS:

type(ESMF_Clock) :: clock
integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if the ESMF object (here clock) is in the created state, .false. otherwise.

3.9.26 NUOPC_StateAdvertiseField - Advertise a Field in a State

INTERFACE:

subroutine NUOPC_StateAdvertiseField(state, StandardName, Units, 
    LongName, ShortName, name, rc)

ARGUMENTS:

type(ESMF_State), intent(inout) :: state
character(*) , intent(in) :: StandardName
character(*), intent(in), optional :: Units
character(*), intent(in), optional :: LongName
character(*), intent(in), optional :: ShortName
character(*) , intent(in), optional :: name
integer, intent(out), optional :: rc

DESCRIPTION:
Advertises a Field in a State. This call checks the provided information against the NUOPC Field Dictionary. Omitted optional information is filled in using defaults out of the NUOPC Field Dictionary.

The arguments are:

state  The ESMF_State object through which the Field is advertised.

StandardName  The StandardName of the advertised Field. Must be a StandardName found in the NUOPC Field Dictionary.

[Units]  The Units of the advertised Field. Must be convertible to the canonical units specified in the NUOPC Field Dictionary for the specified StandardName. If omitted, the default is to use the canonical units associated with the StandardName in the NUOPC Field Dictionary.

[LongName]  The LongName of the advertised Field. NUOPC does not restrict the value of this variable. If omitted, the default is to use the LongName associated with the StandardName in the NUOPC Field Dictionary.

[ShortName]  The ShortName of the advertised Field. NUOPC does not restrict the value of this variable. If omitted, the default is to use the ShortName associated with the StandardName in the NUOPC Field Dictionary.

[name]  The actual name of the advertised Field by which it is accessed in the State object. NUOPC does not restrict the value of this variable. If omitted, the default is to use the value of the ShortName.

[rc]  Return code; equals ESMF_SUCCESS if there are no errors.

3.9.27  NUOPC_StateBuildStdList - Build lists of Field information from a State

INTERFACE:

recursive subroutine NUOPC_StateBuildStdList(state, stdAttrNameList, stdItemNameList, stdConnectedList, stdFieldList, rc)

ARGUMENTS:

type(ESMF_State),  intent(in)  :: state
classic(ESMF_MAXSTR),  pointer  :: stdAttrNameList(:)
classic(ESMF_MAXSTR),  pointer,  optional  :: stdItemNameList(:)
classic(ESMF_MAXSTR),  pointer,  optional  :: stdConnectedList(:)
type(ESMF_Field),  pointer,  optional  :: stdFieldList(:)
integer,  intent(out),  optional  :: rc

DESCRIPTION:

Constructs lists containing the StandardName, Field name, and connected status of the Fields in the state. Returns this information in the list arguments. Recursively parses through nested States.

All pointer arguments present must enter this method unassociated. On return, the deallocation of an associated pointer becomes the user responsibility.
3.9.28 NUOPC_StateIsAllConnected - Check if all the Fields in a State are connected

INTERFACE:

    function NUOPC_StateIsAllConnected(state, rc)

RETURN VALUE:

    logical :: NUOPC_StateIsAllConnected

ARGUMENTS:

    type(ESMF_State), intent(in) :: state
    integer, intent(out), optional :: rc

DESCRIPTION:

    Returns .true. if all the Fields in state are connected. Otherwise returns .false..

3.9.29 NUOPC_StateIsAtTime - Check if all the Fields in a State are at the given Time

INTERFACE:

    function NUOPC_StateIsAtTime(state, time, rc)

RETURN VALUE:

    logical :: NUOPC_StateIsAtTime

ARGUMENTS:

    type(ESMF_State), intent(in) :: state
    type(ESMF_Time), intent(in) :: time
    integer, intent(out), optional :: rc

DESCRIPTION:

    Returns .true. if all the Fields in state have a timestamp that matches time. Otherwise returns .false..

3.9.30 NUOPC_StateIsFieldConnected - Test if Field in a State is connected

INTERFACE:
function NUOPC_StateIsFieldConnected(state, fieldName, rc)

RETURN VALUE:

logical :: NUOPC_StateIsFieldConnected

ARGUMENTS:

  type(ESMF_State), intent(in) :: state
  character(*) , intent(in) :: fieldName
  integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if Fields with name fieldName contained in state is connected. Otherwise returns .false..

3.9.31 NUOPC_StateIsUpdated - Check if all the Fields in a State are marked as updated

INTERFACE:

function NUOPC_StateIsUpdated(state, count, rc)

RETURN VALUE:

logical :: NUOPC_StateIsUpdated

ARGUMENTS:

  type(ESMF_State), intent(in) :: state
  integer, intent(out), optional :: count
  integer, intent(out), optional :: rc

DESCRIPTION:

Returns .true. if all the Fields in state have their "Updated" Attribute set to "true". Otherwise returns .false.. The count argument returns how many of the Fields have the Updated" Attribute set to "true".

3.9.32 NUOPC_StateRealizeField - Realize a previously advertised Field in a State

INTERFACE:

subroutine NUOPC_StateRealizeField(state, field, rc)
ARGUMENTS:

- type(ESMF_State), intent(inout) :: state
- type(ESMF_Field), intent(in) :: field
- integer, intent(out), optional :: rc

DESCRIPTION:

Realizes a previously advertised Field in state.

3.9.33 NUOPC_StateSetTimestamp - Set a time stamp on all Fields in a State

INTERFACE:

subroutine NUOPC_StateSetTimestamp(state, clock, selective, rc)

ARGUMENTS:

- type(ESMF_State), intent(inout) :: state
- type(ESMF_Clock), intent(in) :: clock
- logical, intent(in), optional :: selective
- integer, intent(out), optional :: rc

DESCRIPTION:

Sets the TimeStamp Attribute according to clock on all the Fields in state.

3.9.34 NUOPC_StateUpdateTimestamp - Update the timestamp on all the Fields in a State

INTERFACE:

subroutine NUOPC_StateUpdateTimestamp(state, rootPet, rc)

ARGUMENTS:

- type(ESMF_State), intent(in) :: state
- integer, intent(in) :: rootPet
- integer, intent(out), optional :: rc

DESCRIPTION:

Updates the TimeStamp Attribute for all the Fields on all the PETs in the current VM to the TimeStamp Attribute held by the Field instance on the rootPet.

44
3.9.35  NUOPC_TimePrint - Formatted print of time information

INTERFACE:

    subroutine NUOPC_TimePrint(time, string, unit, rc)

ARGUMENTS:

    type(ESMF_Time), intent(in) :: time
    character(*), intent(in), optional :: string
    character(*), intent(out), optional :: unit
    integer, intent(out), optional :: rc

DESCRIPTION:

Write a formatted time with or without string to unit. If unit is present it must be an internal unit, i.e. a string variable. If unit is not present then the output is written to the default external unit (typically that would be stdout).