

The Cupid Integrated Development Environment for
Earth System Models

Feature Overview and Tutorial
Version 0.1b¹

Rocky Dunlap
NOAA/CIRES/University of Colorado
`rocky.dunlap@noaa.gov`

January 7, 2015



¹This work supported by the NASA CMAC program.

Contents

Contents	2
1 Overview of Features	3
2 ESMF and NUOPC	7
2.1 The Earth System Modeling Framework	7
2.2 The National Unified Operational Prediction Capability	7
3 Tutorial	10
3.1 Installation	10
3.2 Acquiring NUOPC Prototype Applications from SourceForge	10
3.3 Reverse Engineer NUOPC Components	12
3.4 Check a Source File for NUOPC Compliance	16
3.5 Generate NUOPC Code Templates	18
4 Cloud-based Training Environment	23
4.1 Creating a new Cupid Training Project	24
4.2 Compile and Run the NUOPC Single Model with Driver training scenario	29

Chapter 1

Overview of Features

Cupid is a set of development tools to facilitate the adoption of geoscience modeling frameworks into new and existing model codebases. It features a framework-aware code editing environment and a cloud-based configuration tool to simplify configuring the compile and execution environment. The target framework is the Earth System Modeling Framework (ESMF) and its interoperability layer called the National Unified Operation Prediction Capability (NUOPC), which is currently being implemented in most major climate and weather models in the US. Cupid tools are intended for model developers who have prior experience with model development workflows, but are new to developing with ESMF and NUOPC. It is also aimed at developers interested in exploring the benefits of using the Eclipse Integrated Development Environment (IDE) for improving development productivity.

Use of modeling frameworks is quickly becoming the norm for both operational and research climate and weather models. Modeling frameworks provide a number of benefits including mechanisms for componentizing complex codebases, functions and data structures for coupling independent models into a single simulation, increased developer productivity through code reuse, improved quality and robustness of features compared with “home grown” solutions, and fast execution via parallel data transfer and interpolation operators.

In a framework-based application, such as a coupled model that uses ESMF/NUOPC, some application behaviors are provided by the framework and some are provided by the application developer. For example, ESMF provides functions for transferring and interpolating field data from one model’s native grid to another model’s native grid. However, ESMF does not prescribe entirely what it means for the model to take a step forward in time since that behavior is application specific. A framework provides a set of abstractions, *framework-provided concepts*, that the developer is required to instantiate and configure in their code. Creating a framework-based application is called *framework completion* because the developer fills in application behaviors not provided by the framework, or specializes behav-

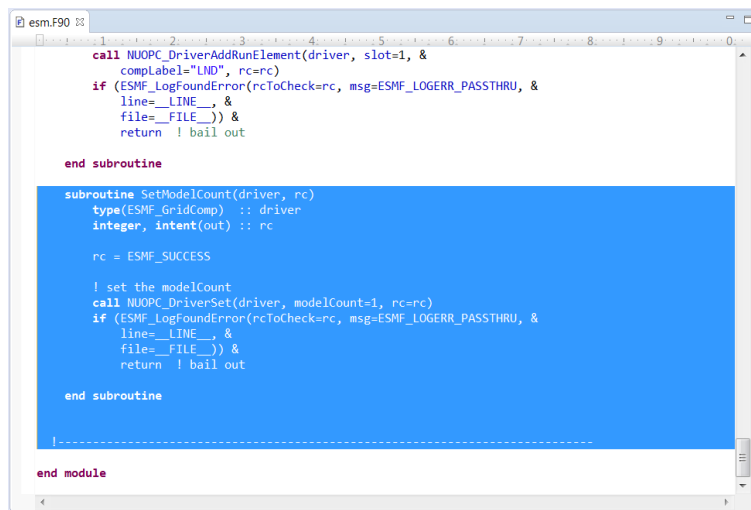
iors provided by the framework. Software engineering research has shown that even for well-designed frameworks, writing correct framework completion code is difficult because it requires a deep understanding of the framework’s behavior.

The Cupid tools adds framework-specific intelligence to the Eclipse Integrated Development Environment in order to facilitate adoption of ESMF and NUOPC. The features include:

- A tool for **reverse engineering** source code to identify how and where the NUOPC API is used. The reverse engineering function does not require execution of the user’s code—instead, it operates during the phase of development when code is written, such as when framework code is first introduced into an existing model. The reverse engineered model is presented to the user alongside the source code in the form of an outline where nodes correspond to NUOPC components, specialization points, or API calls. Clicking on a node brings up the relevant code fragments in the code editor. The outline view also provides contextual technical documentation from the NUOPC reference manual. The reverse engineering tool provides basic checks for code-level compliance to NUOPC technical rules and highlights issues in the NUOPC outline view.

NUOPC Definition	Value
NUOPC Driver	ESM (esm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Driver
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
SetModelServices	SetModelServices
SetModelCount	Registration
SetModelPetLists [0..1]	
SetRunSequence [0..1]	SetRunSequence
New Run Sequence	
Add Run Element [1..n]	"ATM" => "OCN"(slot=1)
Add Run Element [1..n]	"OCN" => "ATM"(slot=1)
Add Run Element [1..n]	"ATM" => "LND"(slot=1)
Add Run Element [1..n]	"LND" => "ATM"(slot=1)
Add Run Element [1..n]	"ATM"(slot=1)
Add Run Element [1..n]	"OCN"(slot=1)
Add Run Element [1..n]	"LND"(slot=1)
Registration	

- A tool for **automatic code generation** of NUOPC-compliant code fragments. The generated code can be used as is, although further customization of the generated code is usually required. The generated code is woven into the user's existing code at the appropriate places, keeping the existing code structure intact. All generated code is highlighted in the editor so the user is aware of what code was generated and can verify its correctness and make any necessary modifications. In some cases default values are chosen for parameters when the value cannot be determined automatically. These parameters are highlighted in a different color to direct the user to them.



```

esm.F90
1      2      3      4      5      6      7      8      9      0
call NUOPC_DriverAddRunElement(driver, slot=1, &
  compLabel="LND", rc=rc)
if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
  line=__LINE__, &
  file=__FILE__)) &
  return ! bail out

end subroutine

subroutine SetModelCount(driver, rc)
  type(ESMF_GridComp) :: driver
  integer, intent(out) :: rc

  rc = ESMF_SUCCESS

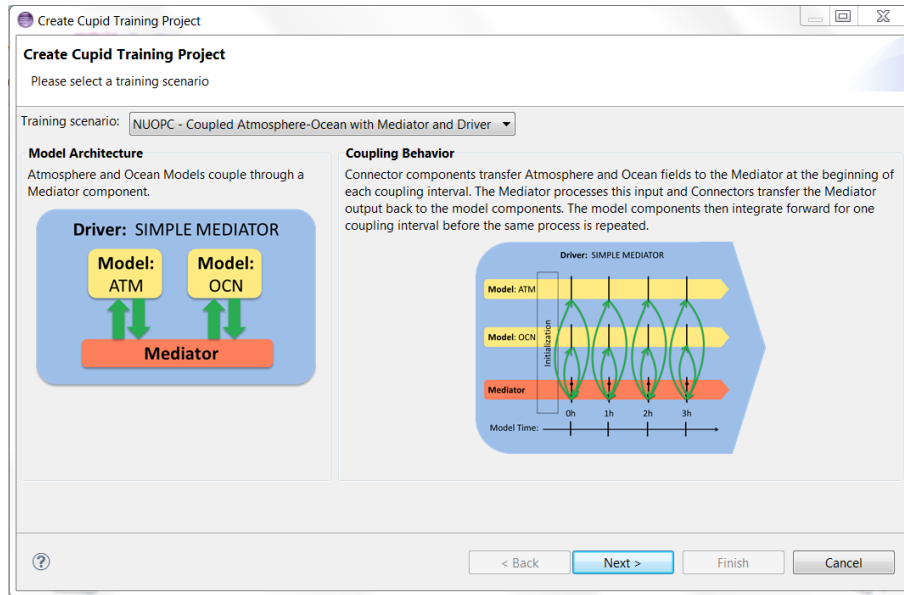
  ! set the modelCount
  call NUOPC_DriverSet(driver, modelCount=1, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=__LINE__, &
    file=__FILE__)) &
    return ! bail out

end subroutine

-----
end module

```

- A **cloud configuration** feature that allows the user to select a training scenario and, within a few minutes, configure, compile, and execute both prototype NUOPC applications on virtual machine instances using the Amazon EC2 platform. *This feature uses the NESII Amazon account, and as such is currently only available for internal use.*



Chapter 2

ESMF and NUOPC

This section describes the Earth System Modeling Framework (ESMF) and the National Unified Operational Prediction Capability (NUOPC) and provides references for those interested in finding out more. Readers already familiar with ESMF and NUOPC may choose to skip this section.

2.1 The Earth System Modeling Framework

ESMF is a high-performance software framework designed for numerical geoscience models. Some of the framework-provided concepts include model components (`ESMF_GridComp`) and coupler components (`ESMF_CplComp`; mediators between model components), and data types for model state (`ESMF_State`), distributed arrays (`ESMF_Array`), physical fields (`ESMF_Field`), and numerical grids (`ESMF_Grid`; discretization schemes). An ESMF-based application is typically designed as a hierarchy of model components where components communicate by exchanging `ESMF_State` objects via framework-provided interfaces. `ESMF_GridComps` and `ESMF_CplComps` have user-customizable `initialize()`, `run()`, and `finalize()` methods. For more information about ESMF, see the ESMF User’s Guide and the ESMF Reference Manual.

2.2 The National Unified Operational Prediction Capability

To promote interoperability of model components, NUOPC is a set of generic components, metadata conventions, and behavioral protocols encoded in a software layer on top of ESMF. Together, these elements form the basis of a *common model architecture*—a standard way of building models in order to make it easier to assemble coupled models using components from different sources. NUOPC is currently being implemented in research and operational models such as the HYCOM ocean

model, GFDL’s MOM5 ocean model, and NASA’s ModelE climate model. Additional information about NUOPC can be found on the NUOPC home page.

NUOPC applications are built by combining four basic building blocks called *generic components*. The four types of generic component are **Driver**, **Model**, **Mediator**, and **Connector**. Many component behaviors have been predefined by NUOPC. However, in some cases, the developer needs to provide implementations of behaviors not defined by NUOPC. Additionally, if the generic behavior does not meet the requirements of the coupled model, the developer may need to override existing behaviors. In both cases, the developer’s implementation is typically provided in subroutines which are registered with and called by the framework. The process of providing new behaviors or overriding existing ones is called *specialization*. As defined here, *specialization* is conceptually similar to how a class overrides a parent class method to provide a different implementation in an object-oriented programming language. However, because the public ESMF and NUOPC APIs are not implemented in an object-oriented language, a custom specialization mechanism has been defined. Understanding the specialization process is essential for adopting NUOPC into a model’s codebase.

The **Driver** generic component implements a harness of ESMF components and ESMF_State objects and it is specialized by plugging in **Model**, **Mediator**, **Connector**, and other **Driver** components. The **Driver** initializes its child components according to an *Initialize Phase Definition* and drives their `run()` methods according to a *Run Sequence*. **Model** wraps a user’s code so it can be plugged into a **Driver**. **Models** represent major geophysical domains such as atmosphere, ocean, and ice. **Connectors** and **Mediators** manage communication between **Models**. **Connectors** implement standard interactions such as parallel redistribution or re-gridding (interpolation) of fields and **Mediators** implement complex **Model** interactions requiring customized code. Figure 2.1 illustrates several possible architectural configurations of NUOPC components.

To take full advantage of NUOPC, developers must ensure that model components comply with NUOPC architectural constraints and technical rules. The full definition of NUOPC compliance is available on the NUOPC compliance web page.

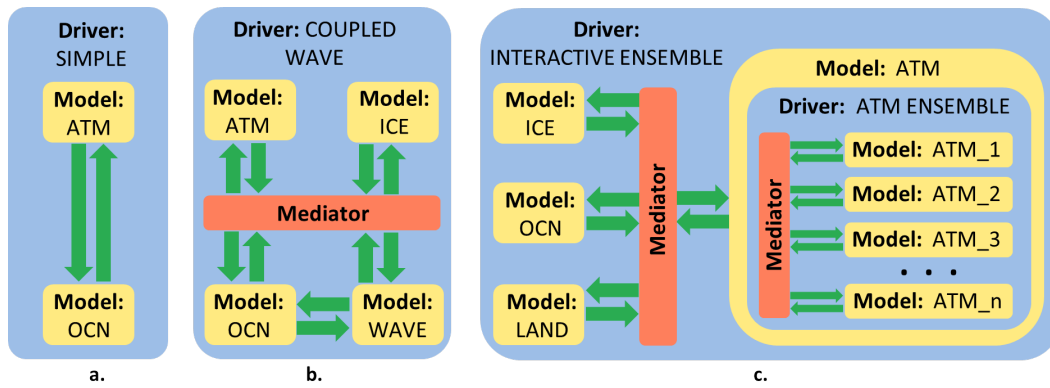


Figure 2.1: **a.** A Driver (blue box) with two child Models (yellow boxes) and simple Connectors (green arrows). **b.** A configuration in which a Mediator (orange box) couples atmosphere, ocean, ice, and wave Models. **c.** A complex configuration showing nested Drivers.

Chapter 3

Tutorial

3.1 Installation

Instructions for downloading and installing Cupid (and the Eclipse IDE itself) are available on the Cupid web site at <https://www.earthsystemcog.org/projects/cupid/>.

3.2 Acquiring NUOPC Prototype Applications from SourceForge

In order to explore Cupid's reverse engineering and code generation capabilities, you need to acquire the source code for at least one of the provided NUOPC prototype applications. A description of the available prototype codes is available at https://earthsystemcog.org/projects/nuopc/proto_codes. The prototype source code itself is available in a Subversion repository on SourceForge:

`http://sourceforge.net/p/esmfcontrib/svn/HEAD/tree/NUOPC/trunk/`

Assuming you have already installed Eclipse and the Cupid plugin, there are a couple ways to acquire the NUOPC prototype code in your local workspace:

1. Download a snapshot from SourceForge and import it into a new Eclipse project.
2. Connect Eclipse to the SourceForge Subversion repository and check out NUOPC prototype code.

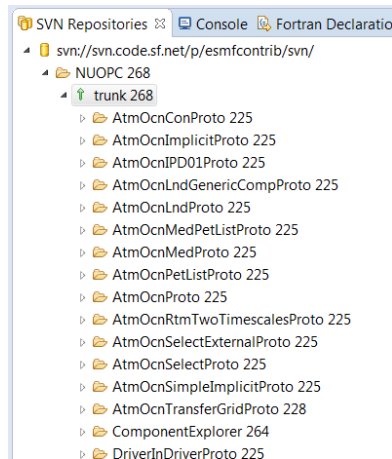
In this section we will describe how to connect Eclipse to the NUOPC prototype Subversion repository and how to check out one of the prototype applications into a new project in your Eclipse workspace.

1. Install the Subversion plugin for Eclipse (Subversive)

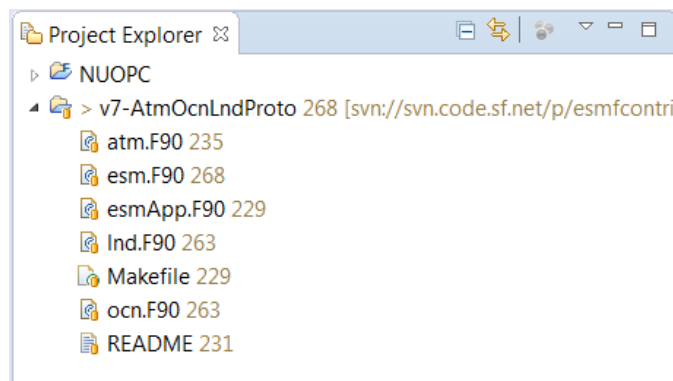
First, ensure that the Subversive plugin, which provides access to Subversion repositories, is installed on your copy of Eclipse. To determine if it is already installed go to Window → Show View → Other... and look for a folder called SVN. If it is there, Subversive is installed and you can skip the rest of this step.

If you need to install Subversive follow these steps:

- a) Select Help → Install New Software from the Eclipse menu
 - b) In the Work With box, choose Luna - <http://download.eclipse.org/releases/luna>
 - c) In the list, choose *Subversive SVN Team Provider* under the *Collaboration* group
 - d) Click Next a couple times, accept the license agreement, and click Finish.
 - e) You will be prompted to restart Eclipse. Choose Yes.
 - f) After Eclipse restarts, choose Window → Show View → Other..., open the SVN folder and choose SVN Repositories
 - g) The first time you select the SVN Repositories view, the *Subversive Connector Discovery* dialog will appear. Select one of the latest SVN connectors (e.g., SVN Kit 1.8.3) and click Finish. Click Next a couple times, accept the license agreement, agree to installing unsigned content, click Finish and restart Eclipse. Subversive is now installed and ready to use.
- ### 2. Add the NUOPC prototype codes repository to Subversive
- a) In the menu, select Window → Show View → Other..., open the SVN folder and choose SVN Repositories.
 - b) In the SVN Repositories view, right click and choose New → Repository Location...
 - c) In the URL field, put the URL of the NUOPC prototype codes Subversion repository: <http://svn.code.sf.net/p/esmfcontrib/svn/> and click Finish. The new repository location now appears in the list.
 - d) In the new repository location, navigate to NUOPC → trunk. Each subfolder under that location is a separate NUOPC prototype application. Right click on the folder *v7-AtmOcnLndProto* and click Check Out.



e) You should now see a new project in your workspace named *v7-AtmOcnLndProto*.

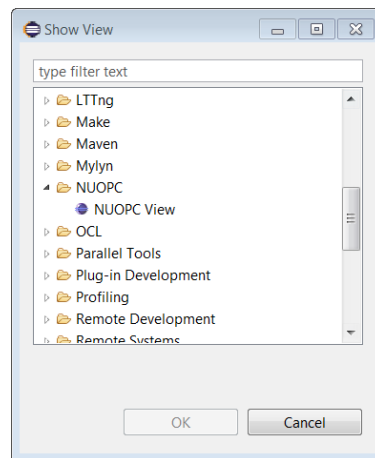


3.3 Reverse Engineer NUOPC Components

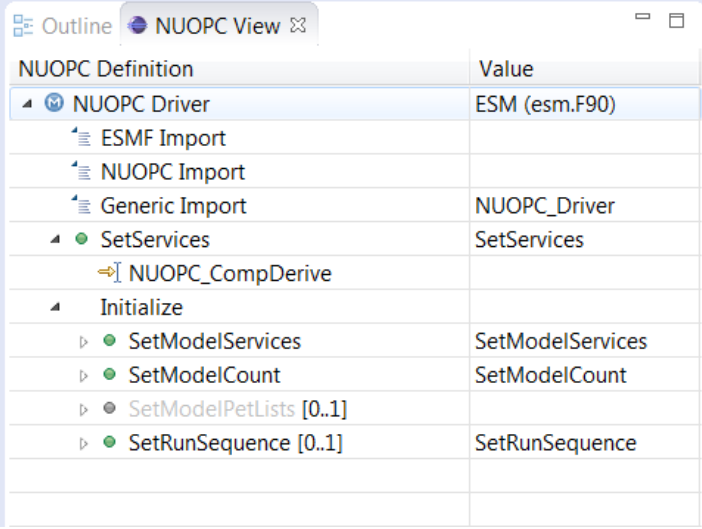
In this section you will learn how use Cupid's reverse engineering function. You should now have a project called *v7-AtmOcnLndProto* in your Eclipse workspace containing several files:

- atm.F90
- esm.F90
- esmApp.F90
- lnd.F90

- Makefile
 - ocn.F90
 - README
1. Open the NUOPC View. From the Eclipse menu, choose Window → Show View → Other... and then select *NUOPC View* from the *NUOPC* folder.

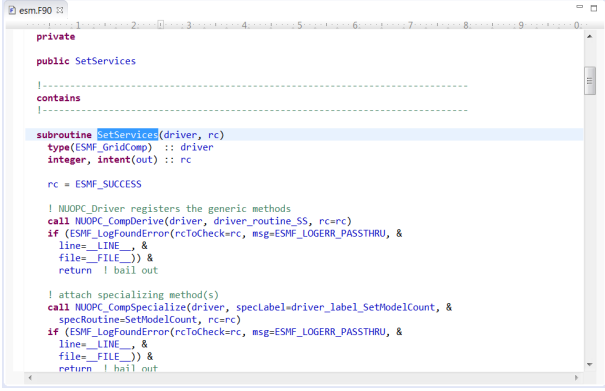


2. Open the file *esm.F90* from the *v7-AtmOcnLndProto* project by double-clicking the file in the Project Explorer. When the NUOPC View is visible, it automatically updates when a new file is opened that contains code for one of the supported NUOPC components (currently Model and Driver). The NUOPC View also updates when a file is saved. *Therefore, after making any modifications to a source file, you must save it first before seeing the updated outline.*
3. The reverse engineered outline now appears in the NUOPC View.



NUOPC Definition	Value
NUOPC Driver	ESM (esm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Driver
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
SetModelServices	SetModelServices
SetModelCount	SetModelCount
SetModelPetLists [0..1]	
SetRunSequence [0..1]	SetRunSequence

- Expand the *NUOPC Driver* element in the NUOPC View (if it's not already expanded) and find the *SetServices* element. The green circle indicates that this element maps to a Fortran subroutine. Double-click the *SetServices* element and the corresponding subroutine name will be highlighted in the source code editor.



```

private
public SetServices
|-----
contains
|-----

subroutine SetServices(driver, rc)
  type(ESMF_GridComp) :: driver
  integer, intent(out) :: rc

  rc = ESMF_SUCCESS

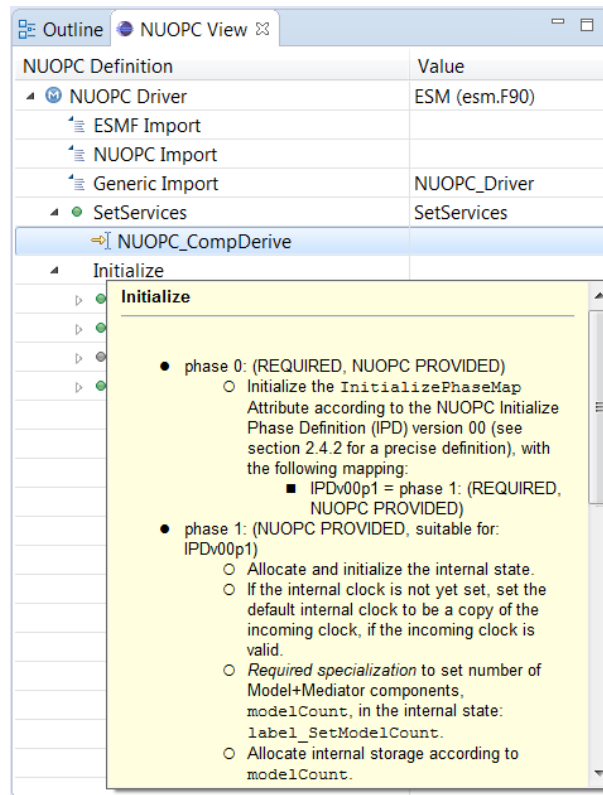
  ! NUOPC_Driver registers the generic methods
  call NUOPC_CompDerive(driver, driver_routine_SS, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=_LINE_, &
    file=_FILE_)) &
    return ! bail out

  ! attach specializing method(s)
  call NUOPC_CompSpecialize(driver, specLabel=driver_label_SetModelCount, &
    specRoutine=SetModelCount, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=_LINE_, &
    file=_FILE_)) &
    return ! bail out
end subroutine SetServices

```

Every NUOPC component is required to have a single public entry point called **SetServices**. This subroutine is called by a parent component or top-level program to register all of the execution phases and specialization points for the component.

5. In the NUOPC View, find the element *NUOPC_CompDerive* under the *SetServices* element. The yellow arrow icon indicates that the element maps to a subroutine call. Double-click the element and the corresponding call will be highlighted in the source code. The call to **NUOPC_CompDerive** indicates that this component derives from (extends) one of the generic NUOPC components, in this case the **NUOPC_Driver** component.
6. In the reverse engineered model, find the element *NUOPC Driver* \rightarrow *Initialize*. Note that it does not have an icon because this element does not map to an element in the reverse engineered code. The NUOPC technical rules defined precisely the Driver initialization process. Hover your mouse over *Initialize* in the NUOPC View to see reference manual documentation about the order of Driver initialization.



Note that the initialize documentation identifies two required specialization points—implementations that the user must provide—**SetModelServices** and **SetModelCount**. These two specialization points have already been imple-

mented in *esm.F90*. Two optional specialization points are also indicated, `SetModelPetLists` and `SetRunSequence`, the latter of which is already implemented in *esm.F90*. Hovering over these specialization points in the NUOPC View will show the reference documentation.

3.4 Check a Source File for NUOPC Compliance

In this section, we will see how the Cupid reverse engineering tool can show compliance issues by identifying required source code elements that are missing. The compliance checking capabilities of Cupid are a companion to two other analysis tools, the *NUOPC Compliance Checker* and *Component Explorer*. The key difference lies in the kinds of analyses performed. At present, Cupid is purely a static analysis tool and identifies compliance issues by examining the abstract syntax of NUOPC application source code. The *NUOPC Compliance Checker* and *Component Explorer* support dynamic analysis, outputting compliance information in log form as the NUOPC application executes. As such these runtime tools can identify a greater number of compliance issues.

However, because Cupid's reverse engineering works on partially completed code, it is useful for identifying many compliance issues earlier in the development process, before any runs are attempted. The results of the static compliance check are available immediately as the NUOPC application is being developed and, as we will see in the next chapter, Cupid's code generation feature provides some additional assistance to the developer in bringing the code to a compliant state. More information about the runtime compliance checking tools can be found in section 5 of the NUOPC Reference Manual.

In the steps that follow, we will create a compliance issue in the *v7-AtmOcnLndProto* application and see the results in the reverse engineered model.

1. Now open the file *atm.F90* by double-clicking it in the Project Explorer. The NUOPC View should now update with content from that file.

NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
Initialize Phase 1	InitializeP1
Initialize Phase 2	InitializeP2
SetClock [0..1]	
DataInitialize [0..1]	
Run	
SetRunClock [0..1]	
CheckImport [0..1]	
ModelAdvance	ModelAdvance
Finalize	
Finalize Phase 1 [0..1]	

The NUOPC View shows that *atm.F90* is a NUOPC Model. Items that are grayed out in the NUOPC View indicate specialization points or other NUOPC concepts that were not found in the code. This does not necessarily indicate a problem, as some specialization points are optional.

2. If a required specialization point is missing, it will be indicated in the outline by red text. You will now modify *atm.F90* to remove a required specialization point. Find the subroutine **ModelAdvance** towards the bottom of the file, highlight the entire subroutine, delete it, and then save the file. The NUOPC View will now update and *ModelAdvance* will appear in red. Undo the delete by selecting Edit → Undo Typing from the Eclipse menu. Save the file again and **ModelAdvance** will again appear in black.

NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
Initialize Phase 1	InitializeP1
Initialize Phase 2	InitializeP2
SetClock [0..1]	
DataInitialize [0..1]	
Run	
SetRunClock [0..1]	
CheckImport [0..1]	
ModelAdvance	
Registration	
Finalize	
Finalize Phase 1 [0..1]	

3.5 Generate NUOPC Code Templates

Even if a software framework is well designed, writing framework completion code is notoriously difficult, even for seasoned developers. Often, completing a single logical task requires making several code additions at multiple places spread throughout the application source code. If one or more of the required additions are inadvertently left out, the application may not behave as expected.

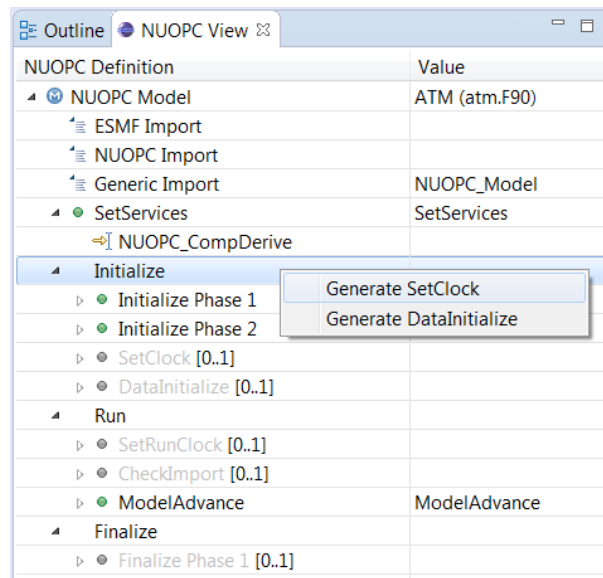
In the software engineering research community, many ideas have been proposed for how to help developers write framework-based applications correctly and efficiently. For ESMF and NUOPC, guidance is provided in the form of comprehensive API documentation (ESMF Reference Manual, NUOPC Reference Manual), system tests (included with source distribution), and small prototype codebases that show how to structure NUOPC applications based on the components in the modeled system (e.g., standalone atmosphere, coupled atmosphere-ocean, three-component system, etc.).

Cupid's code generation feature complements these static resources by generating on-the-fly NUOPC-compliant source code fragments directly in existing source files. The user initiates a code fragment generation by adding elements to a reverse engineered model in the NUOPC View. The generated code fragments can then be customized by the developer.

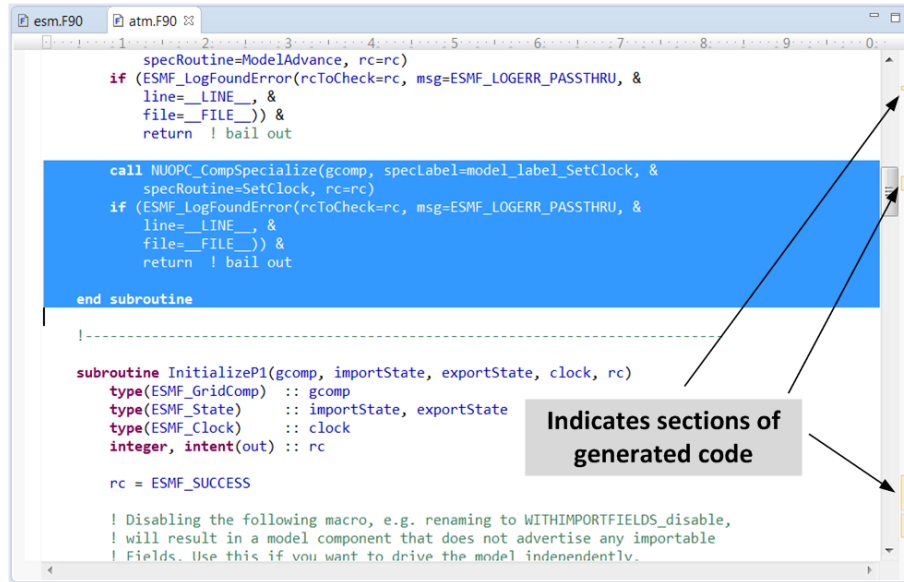
In this section we will demonstrate the code generation capabilities of Cupid by

generating a subroutine stub for an optional specialization point and registering the subroutine in the `SetServices` method of the model.

1. Make sure the file *atm.F90* from the project *v7-AtmOcnLndProto* is open and you can see the reverse engineered outline in the NUOPC View.
2. Notice that the *SetClock* element under *Initialize* is grayed out because that optional specialization point has not been implemented in this file. Right click (Ctrl + Click on Mac) on *Initialize* to bring up the context menu. Click *Generate SetClock*.



3. Both the NUOPC View and the code in the editor have now been updated with the generated code. The yellow bars on the small vertical bar directly to the right of editor indicate which blocks of code were generated by Cupid. Clicking on a yellow block highlights the generated code, which has been inserted into the existing source file. A vertical yellow line also appears immediately to the left of generated code fragments. The generated code is shown in listings 3.1, 3.2, and 3.3.



Listing 3.1: An updated Use statement with the import of `label_SetClock` added (line 4).

```

1 use NUOPC_Model, only: &
2   model_routine_SS => routine_SetServices, &
3   model_label_Advance => label_Advance, &
4   model_label_SetClock => label_SetClock

```

Listing 3.2: A call to register the generated subroutine as the `SetClock` specialization point. This call is inserted at the end of the `SetServices` method.

```

1 call NUOPC_CompSpecialize(gcomp, specLabel=model_label_SetClock, &
2   specRoutine=SetClock, rc=rc)
3 if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
4   line=__LINE__, &
5   file=__FILE__)) &
6   return ! bail out

```

Listing 3.3: A subroutine template that implements the `SetClock` specialization point. The subroutine is inserted at the end of the module.

```

1 subroutine SetClock(gcomp, rc)
2   type(ESMF_GridComp) :: gcomp

```

```

3      integer, intent(out) :: rc
4
5      ! local variables
6      type(ESMF_Clock) :: clock
7      type(ESMF_TimeInterval) :: stabilityTimeStep
8
9      rc = ESMF_SUCCESS
10
11     ! query the Component for its clock, importState and exportState
12     call ESMF_GridCompGet(gcomp, clock=clock, rc=rc)
13     if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
14         line=__LINE__, &
15         file=__FILE__)) &
16         return ! bail out
17
18     ! initialize internal clock
19     ! here: parent Clock and stability timeStep determine actual model timeStep
20     call ESMF_TimeIntervalSet(stabilityTimeStep, m=5, rc=rc)
21     if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
22         line=__LINE__, &
23         file=__FILE__)) &
24         return ! bail out
25
26     call NUOPC_CompSetClock(gcomp, clock, stabilityTimeStep, rc=rc)
27         if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
28             line=__LINE__, &
29             file=__FILE__)) &
30             return ! bail out
31
32 end subroutine

```

4. In some cases default or representative values are selected for parameters to API calls. These parameters are highlighted in blue to draw your attention since they will likely need to be changed. In the `SetClock` subroutine generated in the previous step, note that one of the parameters to the `ESMF_TimeIntervalSet` has been highlighted. The locations of defaulted parameters are also indicated on the vertical bar to the right of the editor.

```
! initialize internal clock
! here: parent Clock and stability timeStep determine actual model timeStep
call ESMF_TimeIntervalSet(stabilityTimeStep, m=5, rc=rc)
if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=__LINE__, &
    file=__FILE__)) &
    return ! bail out
```

Parameters set to default
values are highlighted with a
blue background

Chapter 4

Cloud-based Training Environment

Note: The cloud-based features of Cupid require access to the NESII Amazon EC2 account. As such, this capability is only available internally.

Cupid simplifies the process of configuring a computational environment capable of compiling and executing high-performance geoscience models. IDEs package a lot of development tools into a single application to help manage and simplify the software development workflow. Although IDEs aim to increase developer productivity, they can still introduce a steep learning curve. Some challenges with using IDEs for geoscience model development include:

- Understanding the basic steps involved in moving from source code to a running model
- Making sense of the many development tools and features available in the IDE
- Setting up a high-performance computational environment capable of configuring, compiling and executing model code
- Configuring the IDE to connect to remote computational environments

Cupid's cloud integration feature allows a developer to select a training scenario and, within a few minutes, configure, compile, execute, and view the standard output of both skeleton models and realistic models. (This does not include data file post processing or plot generation.) This feature relies on a set of pre-configured machine images that can be instantiated on Amazon EC2 cloud infrastructure. The machine images contain all of the necessary software dependencies for the selected

scenario. Furthermore, Cupid automatically configures the IDE to connect to and synchronize source code with cloud-based virtual machines.

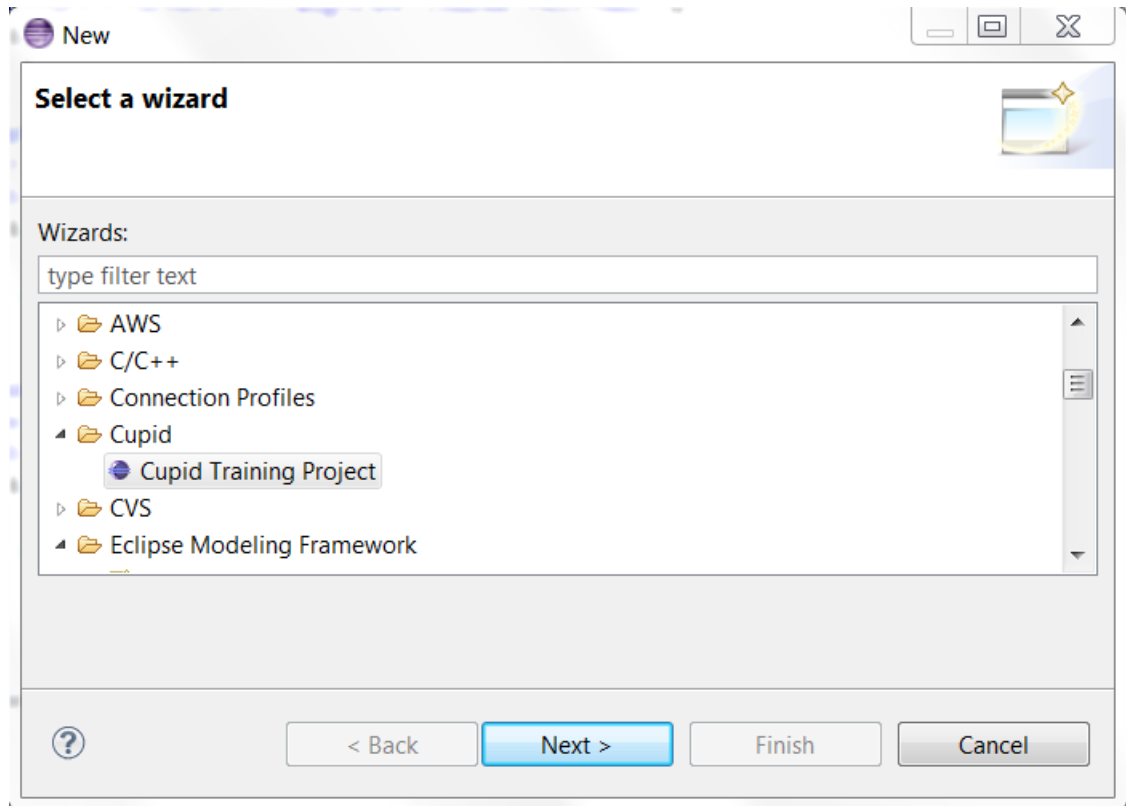
While understanding how to set up and use a high-performance computational environment, including acquiring and configuring prerequisite software packages, is an important part of geoscience modeling, it should not preclude non-experts from quickly setting up a complete computational environment and begin getting their feet wet with writing framework-based code.

4.1 Creating a new Cupid Training Project

This section of the tutorial describes how to use the *New Cupid Training Project Wizard* to create a new Eclipse project, populate it with a skeleton NUOPC application, and start up a remote virtual machine instance for compiling and executing the project.

Before beginning this section, please ensure that your Amazon EC2 credentials have been set up in the Cupid preferences. For instruction on how to do this, see section ??.

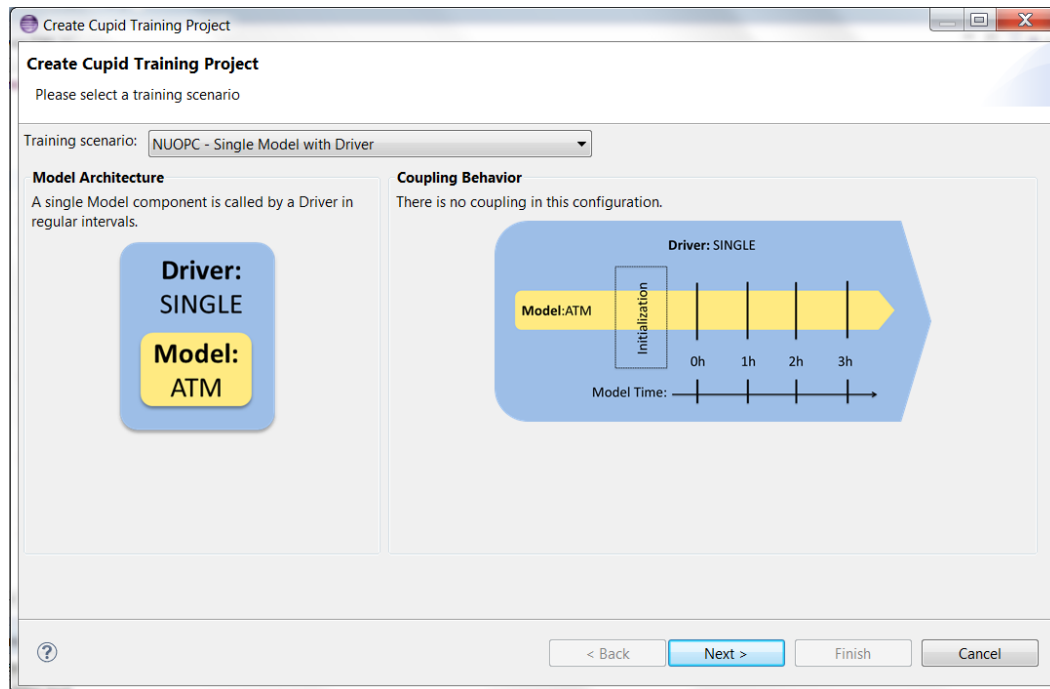
1. **Click File → New → Other and choose Cupid Training Project in the Cupid folder. Click Next.**



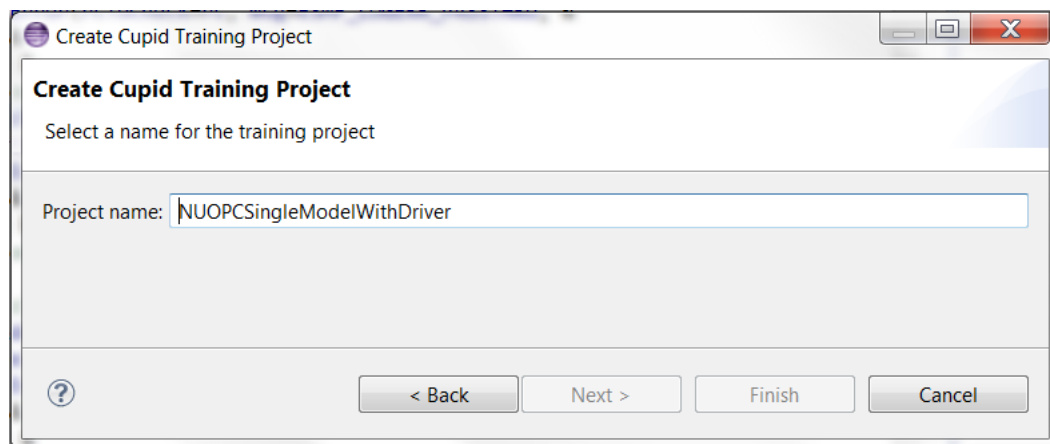
2. On the first page of the wizard, choose a training scenario.

- NUOPC — Single Model with Driver
- NUOPC — Coupled Atmosphere-Ocean Driver
- NUOPC — Coupled Atmosphere-Ocean with Mediator and Driver
- ModelE — Basic Configuration (EM20 rundeck) (*coming soon*)

Changing the training scenario in the list will update the screen to show the model architecture and coupling behavior of the selected scenario. **Choose the scenario “NUOPC — Single Model with Driver” and click Next.**



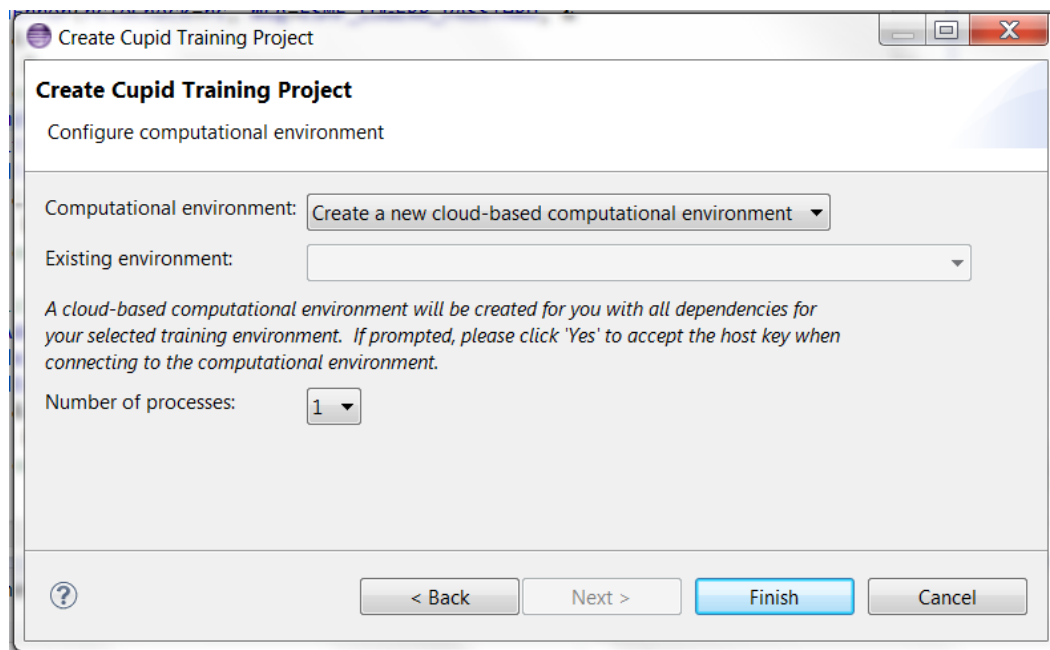
3. **Choose a name for your project or accept the default name.** Note that project names must be unique, so you must choose a name that does not already exist in your workspace. **Click Next.**



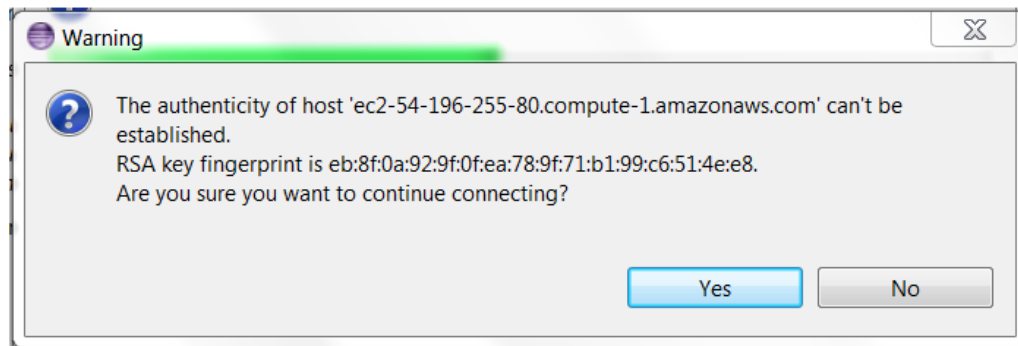
4. Choose a computational environment on which to compile and run the training scenario. The three options are:
 - Create a cloud-based computational environment
 - Use my local machine
 - Use an existing remote environment (*this option is not always available*)

The first option will create a preconfigured virtual machine instance for you with all dependent software (model source code, Fortran compiler, MPI, NetCDF, ESMF, etc.). This is the recommended option unless you know you have a supported environment already set up locally. In some cases, a third option will be available to re-use an existing remote environment that is already running.

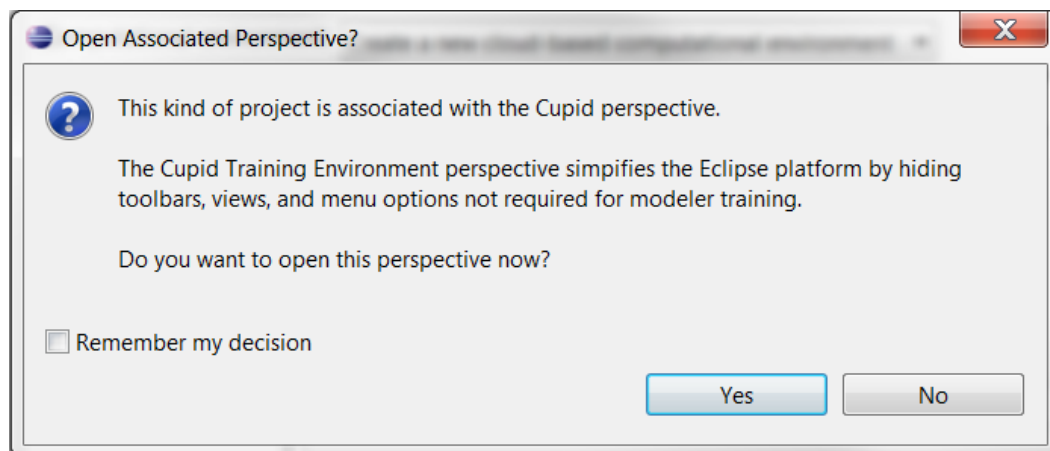
Choose the first option and leave the number of processes at 1. Click Finish.



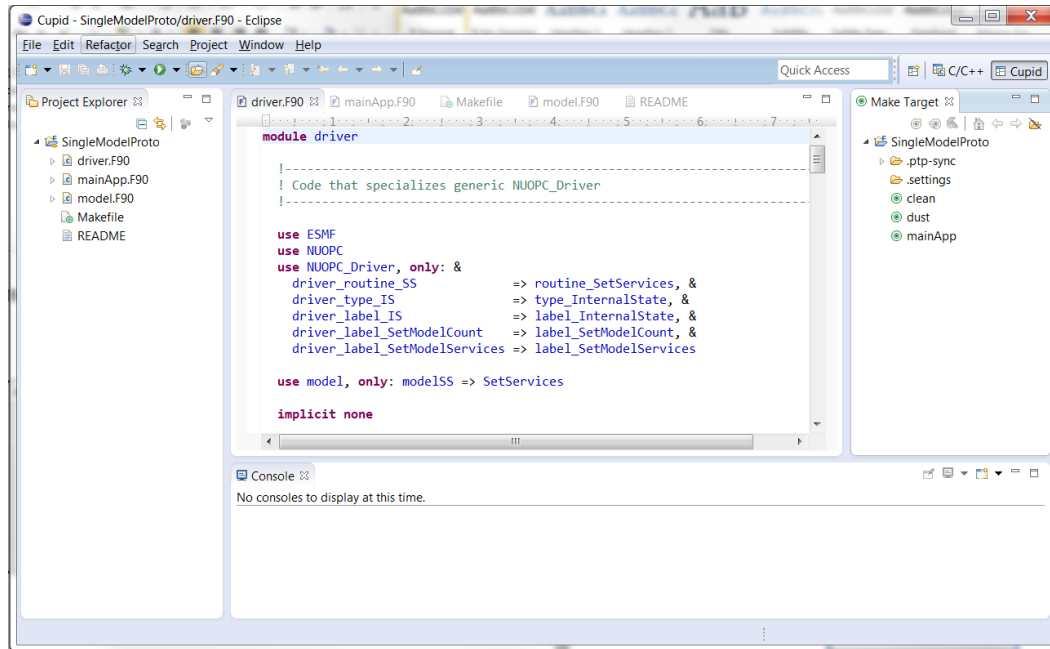
5. It will take up to several minutes for the new computational environment to start up. **During this process you will be asked to accept the host SSH key. Click Yes.**



You might also be asked if you would like to switch to the Cupid Perspective. An Eclipse perspective is a particular screen layout customized for specific tasks. The Cupid Perspective hides a number of Eclipse tools and commands that are not required for the training. **Click Yes to switch to the Cupid Perspective.**



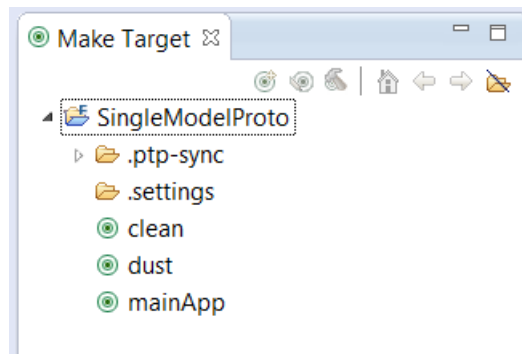
6. You should now see your project in the Project Explorer on the left. The Cupid Perspective also exposes several other views including a Fortran source code editor, a Make Target view on the right for compiling your code and a Console at the bottom for viewing output from the compiler.



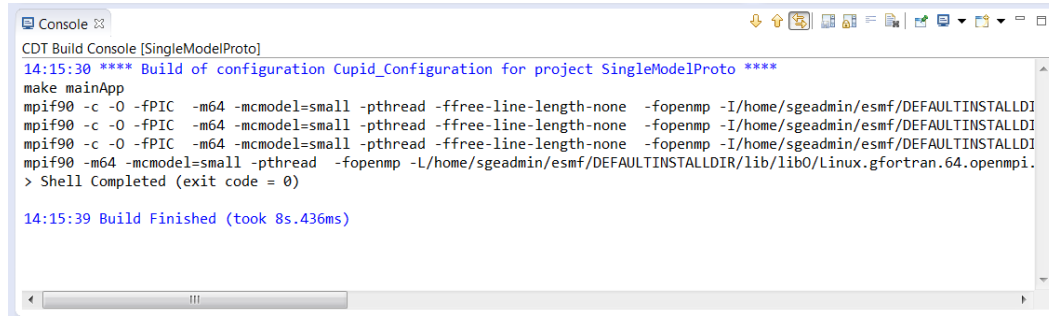
4.2 Compile and Run the NUOPC Single Model with Driver training scenario

In this section of the tutorial, you will learn how to compile and execute the source code provided in the NUOPC Single Model with Driver training scenario. An Eclipse Cheat Sheet is available for this task. (Click Help → Cheat Sheets and choose “Compile and Run a NUOPC Application” in the Cupid folder.)

1. Ensure that the Make Target view is showing (see below). If not, **choose Window → Show View → Other... and select “Make Target” under the Make folder.** Open the folder in the Make Target view with the same name as the project you created.



- If you used the New Cupid Training Project wizard to set up the training scenario, then the correct make targets have already been set up. To compile the NUOPC application, double click the “mainApp” target. You should be able to see the compiler output in the Console view at the bottom of the screen.



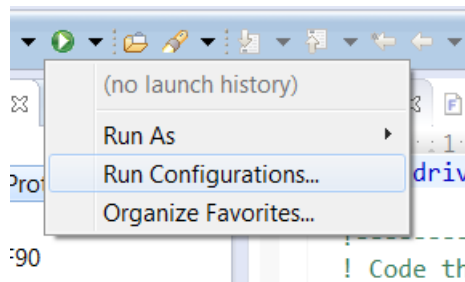
```

CDT Build Console [SingleModelProto]
14:15:30 **** Build of configuration Cupid_Configuration for project SingleModelProto ****
make mainApp
mpif90 -c -O -fPIC -m64 -mcmodel=small -pthread -ffree-line-length-none -fopenmp -I/home/sgeadmin/esmf/DEFAULTINSTALLDIR/lib/lib0/Linux.gfortran.64.openmpi.
mpif90 -c -O -fPIC -m64 -mcmodel=small -pthread -ffree-line-length-none -fopenmp -I/home/sgeadmin/esmf/DEFAULTINSTALLDIR/lib/lib0/Linux.gfortran.64.openmpi.
mpif90 -c -O -fPIC -m64 -mcmodel=small -pthread -ffree-line-length-none -fopenmp -I/home/sgeadmin/esmf/DEFAULTINSTALLDIR/lib/lib0/Linux.gfortran.64.openmpi.
mpif90 -m64 -mcmodel=small -pthread -fopenmp -L/home/sgeadmin/esmf/DEFAULTINSTALLDIR/lib/lib0/Linux.gfortran.64.openmpi.
> Shell Completed (exit code = 0)

14:15:39 Build Finished (took 8s.436ms)

```

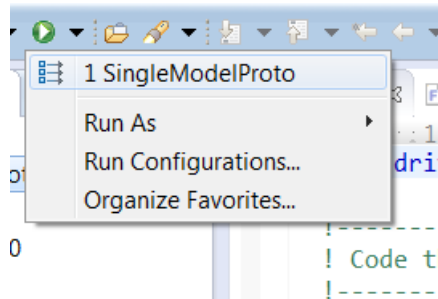
- To run the compiled code, you must set up a run configuration. (*Note: This step will be automated in a future release of Cupid.*) Click the down arrow next to the Run As... button in the toolbar (green circle with a white arrow) and choose Run Configurations from the popup menu.



- In the Run Configurations dialog, right click on Parallel Application and choose “New.” Configure the run configuration as follows:

- Resources tab
 - Target System Configuration: **Open MPI-Generic-Interactive**
 - Connection type: **Remote**
 - Connection: **Cupid Environment (Amazon EC2 ;Project Name;)**
- Application tab
 - Project: should default to your project name
 - Application program: Click Browse and select “mainApp”. The final path should be **/home/sgeadmin/SingleModelProto/mainApp**
- Click **Apply** then **Run**

- Once the Run Configuration has been created, you do not need to set it up again unless you need to change configuration settings. After running it the first time, the run configuration should be available in the Run Configurations dropdown list on the toolbar.



- Standard output from the run will be shown in the Console view.

