Cupid Documentation

Release 2.0

Rocky Dunlap

Oct 19, 2018

Contents

Contents:

CHAPTER 1

Overview

Cupid is a plugin for the Eclipse Integrated Development Environment (IDE) and provides development tools for building and analyzing applications based on the National Unified Operational Prediction Capability (NUOPC) software Layer. The main features include analysis of traces for debugging and profiling coupled systems, and generation of NUOPC-compliant code.

1.1 Key Features

Cupid is as a *framework-aware* development environment with a set of tools specific to ESMF and NUOPC. The key features include:

- Tools to **profile and debug NUOPC applications** through trace analysis and visualization. ESMF has a built-in tracing capability that automatically instruments model components to collect timing, memory, and component metadata. Once a trace has been generated, it can be imported into Cupid for analysis of component execution times, timing of user-defined regions, load balancing, and basic memory usage.
- A **reverse engineering engine** that reads existing NUOPC cap code and presents relevant initialize, run, finalize phases and specialization points in an outline view. The outline is synchronized automatically as the code changes. The tool indicates code-level compliance issues that may result in runtime errors. (The compliance checking is limited to code errors than can be determined by static analysis.)
- A **code generation engine** that outputs NUOPC-compliant code fragments (i.e., initialization phases and specialization points). The generated code can often be used as is, although further customization of the generated code is supported. The generated code is inserted into the user's existing code at the appropriate places, keeping the existing code structure intact. The code generation feature helps the developer understand what framework code is required and where it should be located.

1.2 What is NUOPC?

NUOPC is a consortium of Navy, NOAA, and Air Force modelers and their research partners. It aims to advance the weather prediction modeling systems used by meteorologists, mission planners, and decision makers. NUOPC partners are working toward a common model architecture - a standard way of building models - in order to make it easier to collaboratively build modeling systems. To this end, they have developed a NUOPC Layer that defines conventions and templates for using the Earth System Modeling Framework (ESMF).

Note: The following resources are a good starting point for learning about the NUOPC Layer.

- The NUOPC home page: https://www.earthsystemcog.org/projects/nuopc
- The NUOPC reference manual and how to guide: https://www.earthsystemcog.org/projects/nuopc/refmans

1.3 What is Eclipse?

Eclipse is a graphical user interface or integrated development environment (IDE) used in computer programming for writing software. It contains a base workspace and an extensible plugin system for customizing the environment. Eclipse supports development in a variety of programming languages through the use of plugins. Many of these are found in the Eclipse Marketplace, a kind of “app store” for Eclipse.

CHAPTER 2

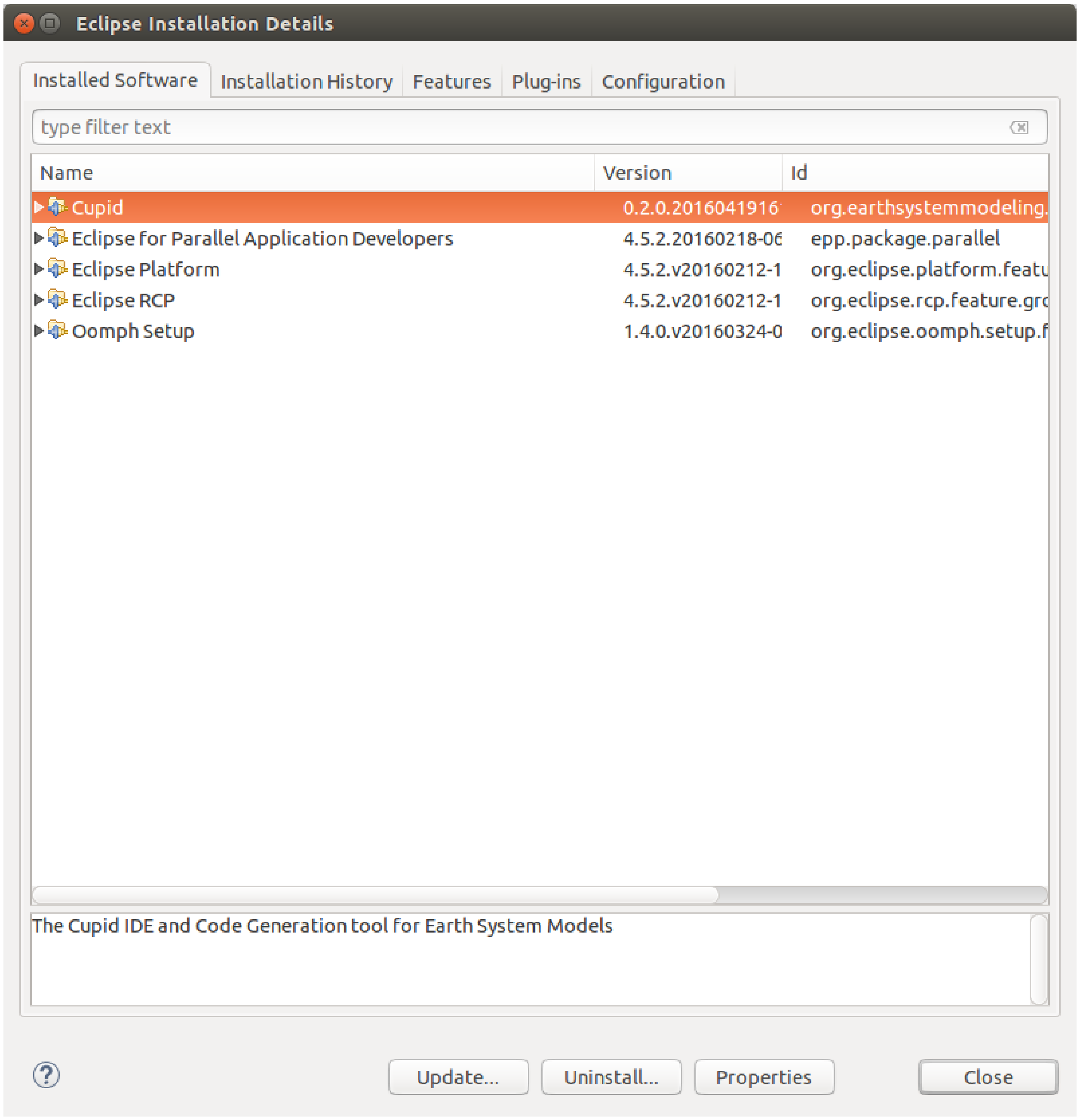
Installation

Cupid is available for installation in the Eclipse Marketplace. To install Cupid, follow these steps:

- If you don't already have it, install Java 8.
- Download and install **Eclipse for Parallel Application Developers** from eclipse.org. You can use the Eclipse Installer which helps to manage multiple installations of Eclipse.
- Start Eclipse and select Help–>Eclipse Marketplace from the menu.
- In the Marketplace window, search for "Cupid". Click the "Install" button to install Cupid. Once installed, you will be prompted to restart Eclipse.

2.1 Verify that Cupid is Installed

To verify that Cupid is installed, view the Eclipse Installation Details by selecting **Help -> About Eclipse** from the Eclipse menu and clicking **Installation Details.** You should see Cupid in the list of installed software.



Name	Version	Id
Cupid	0.2.0.2016041916	org.earthsystemmodeling.
Eclipse for Parallel Application Developers	4.5.2.20160218-06	epp.package.parallel
Eclipse Platform	4.5.2.v20160212-1	org.eclipse.platform.featu
Eclipse RCP	4.5.2.v20160212-1	org.eclipse.rcp.feature.grc
Oomph Setup	1.4.0.v20160324-0	org.eclipse.oomph.setup.f

The Cupid IDE and Code Generation tool for Earth System Models

CHAPTER 3

NUOPC Trace Analysis

Important: These features require a new tracing capability available in a development snapshot of ESMF. **You must use at least ESMF 7.1.0 beta snapshot 31.**

The following command can be used to acquire this development snapshot of ESMF:

git archive –remote=git://git.code.sf.net/p/esmf/esmf –format=tar –prefix=esmf/ ESMF_7_1_0_beta_snapshot_31 | *tar xf -*

To check out other snapshots, simply replace the *ESMF_7_1_0_beta_snapshot_31* with another tag.

Development snapshots are built in the same way as releases. Development snapshots are not formal releases of ESMF and are "use at your own risk." Efforts are made to ensure that most unit and system tests are passing on typical platforms, but there are no guarantees of the stability of development snapshots.

For more information about development snapshots of ESMF, please email: esmf_support@list.woc.noaa.gov.

This section describes how to use Cupid's features for analyzing a NUOPC application trace. The analyses show component timings and profiles and is useful for discovering expensive execution phases and determining if there is a load imbalance in the coupled system. The analyses are custom analyses based on the TraceCompass Eclipse plugin. This plugin will automatically install with Cupid.

The overall process requires two steps:

- First, execute a NUOPC application with tracing turned on
- Second, import the trace into Cupid for analysis and visualization

3.1 Generate a Trace of a NUOPC Application

First, make sure you compile your application with **ESMF version 7.1 beta snapshot 31** or later.

Then, set the following environment variable before the run:

```
$ export ESMF_RUNTIME_TRACE=ON
```

This will automatically instrument the components to collect timing and other information to put into the trace.

Though not required, it is *highly recommended* to set the environment variable *ESMF_RUNTIME_TRACE_PETLIST* to limit which PETs are traced. This will help to limit the size of the output trace and speed up the analysis. If you do not set this environment variable, all PETs will be traced by default. Each PET should be separated by a space, and you can use the notation "X-Y" to indicate a range of PETs. **A good approach is to trace only the root PET of each component in the NUOPC application.** Note that PET zero will always be traced, regardless of the *ESMF_RUNTIME_TRACE_PETLIST* setting.

```
# turn on tracing for PETs 0, 32, and 64 through 72
$ export ESMF_RUNTIME_TRACE_PETLIST="0 32 64-72"
```

After setting these environment variables execute the NUOPC application in the way you normally do. The trace itself will be placed into the *traceout* directory. The directory will contain a *metadata* file and one file per PET that was traced. For example, if PETs 0, 144, and 168 having tracing enabled, the *traceout* directory looks like this:

```
[Rocky.Dunlap@tfe04 traceout]$ ls -la
total 2320
drwx--S--- 2 Rocky.Dunlap stmp    4096 Apr 11 22:20 .
drwxr-sr-x 5 Rocky.Dunlap stmp   73728 Apr 11 23:20 ..
-rw-r----- 1 Rocky.Dunlap stmp 1048576 Apr 11 22:41 esmf_stream_0
-rw-r----- 1 Rocky.Dunlap stmp  229376 Apr 11 22:41 esmf_stream_144
-rw-r----- 1 Rocky.Dunlap stmp  163840 Apr 11 22:41 esmf_stream_168
-rw-r----- 1 Rocky.Dunlap stmp    3370 Apr 11 22:41 metadata
```

The trace files are in a binary format called Common Trace Format so they cannot be viewed directly. If the run was performed on a remote machine, the trace directory needs to be transferred to your local machine where Eclipse is installed. Tar the entire directory and copy it to your machine.

```
$ tar cfz traceout.tar.gz traceout
# scp traceout.tar.gz to your local machine where Eclipse is installed
```

3.2 Import and Open the Trace

3.2.1 Import the Trace

In Eclipse, choose "File -> Import…" from the menu and select "Trace Import" in the folder "Tracing Project."

Click Next. On the next screen select the trace to import. You can import a trace by either selecting the root directory of the trace or by selecting an archive file containing the trace directory. After selecting the root directory or archive, check the trace root folder in the list (see figure below). Then click Finish.

When complete, you will see a new project in the Project Explorer called *Tracing* with a folder called *Traces*. This folder contains the imported trace. It will have a name that matches the archive file or root directory you selected. Double-click to open the trace and see the list of trace events.

If you already have an existing tracing project set in Eclipse, you can add traces to it by right-clicking on the Traces folder and selecting Import from the context menu.

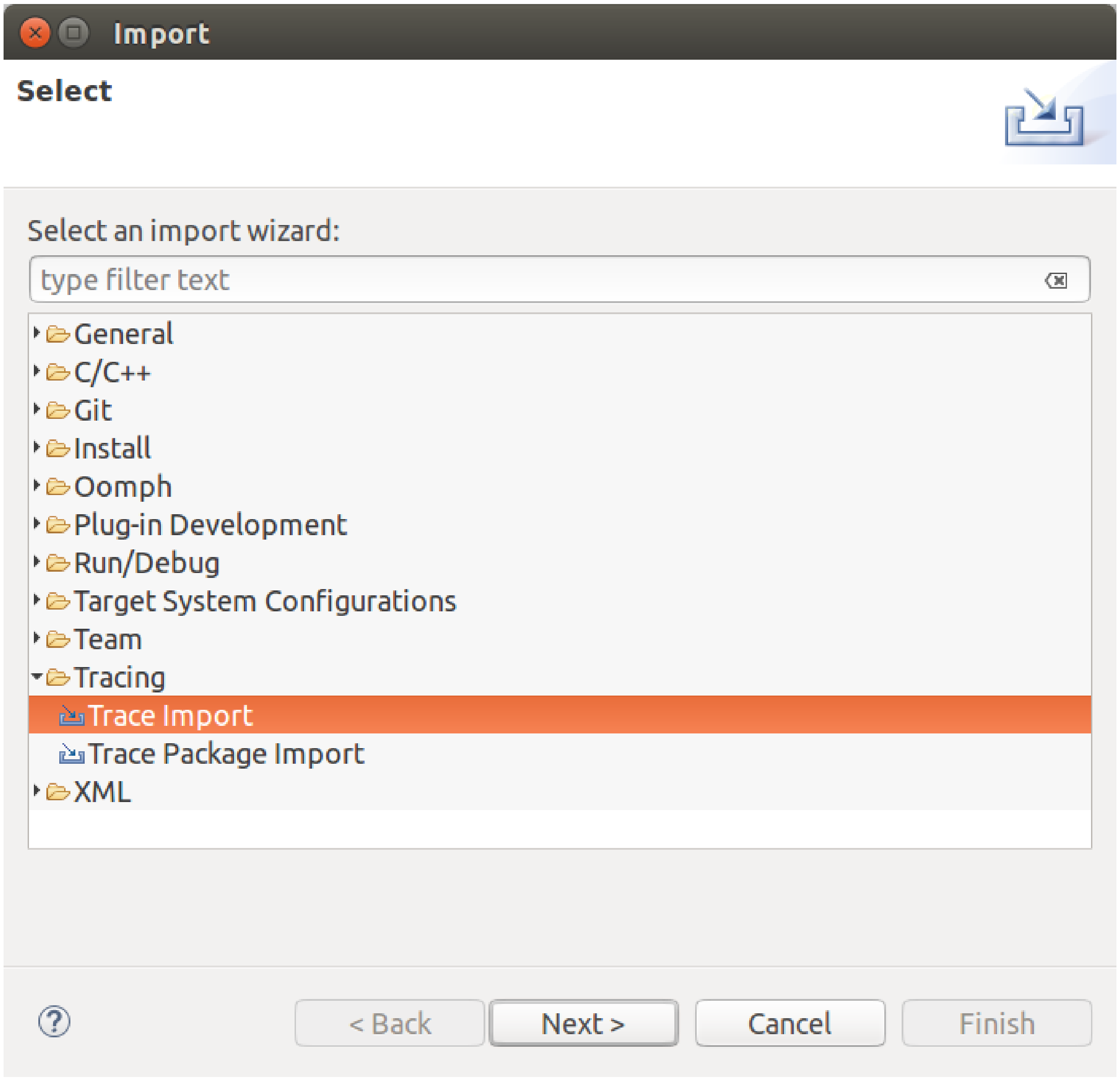
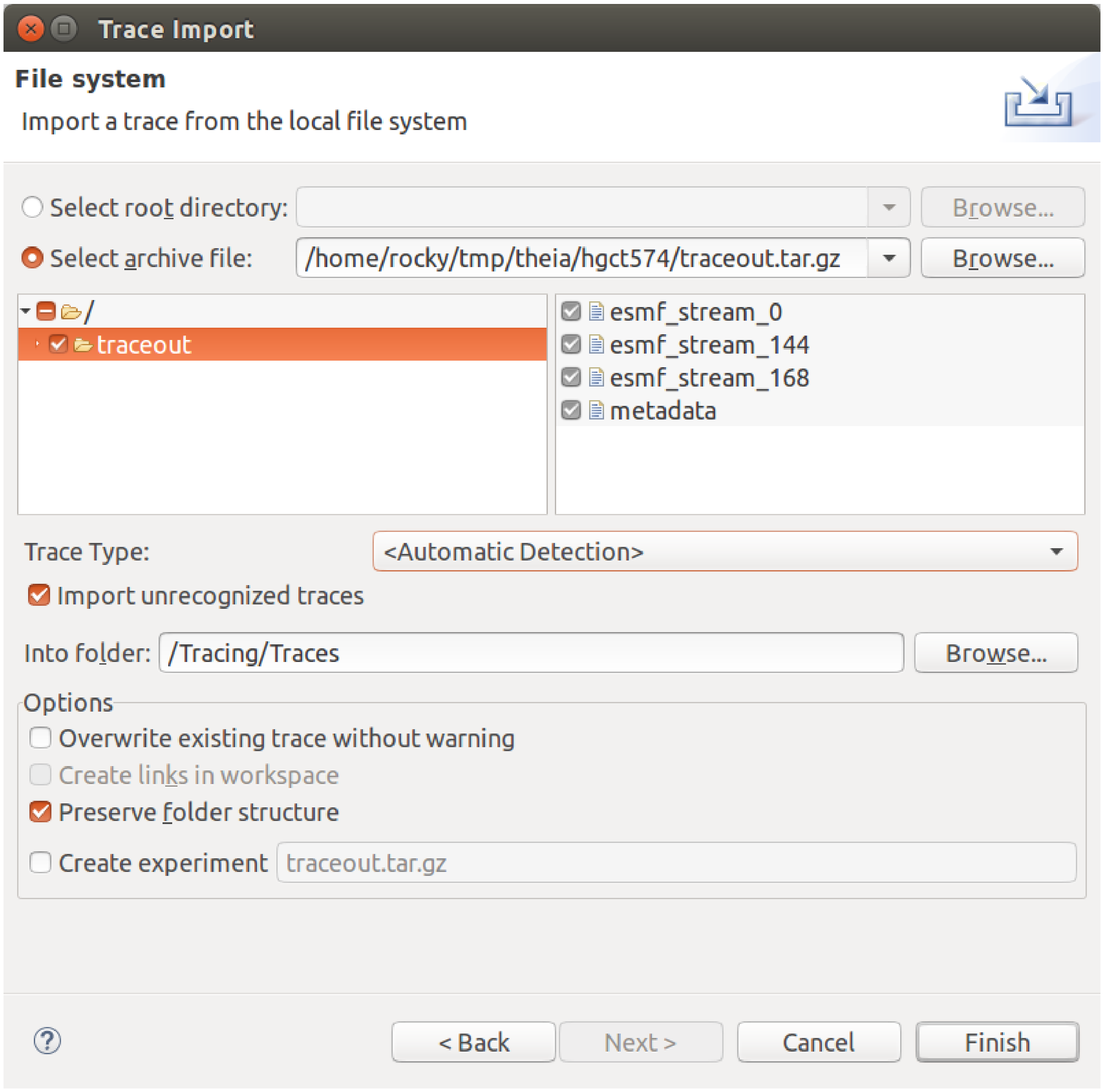


Fig. 1: Import a trace into Eclipse


Trace Import

File system

Import a trace from the local file system

Select root directory: [✓ Select archive file:] /home/rocky/tmp/theia/hgct574/traceout.tar.gz

/
traceout ✓

esmf_stream_0 ✓
esmf_stream_144 ✓
esmf_stream_168 ✓
metadata ✓

Trace Type: <Automatic Detection>

✓ Import unrecognized traces

Into folder: /Tracing/Traces

Options

☐ Overwrite existing trace without warning
☐ Create links in workspace
✓ Preserve folder structure
☐ Create experiment traceout.tar.gz

Fig. 2: Select trace root directory or trace archive file to import

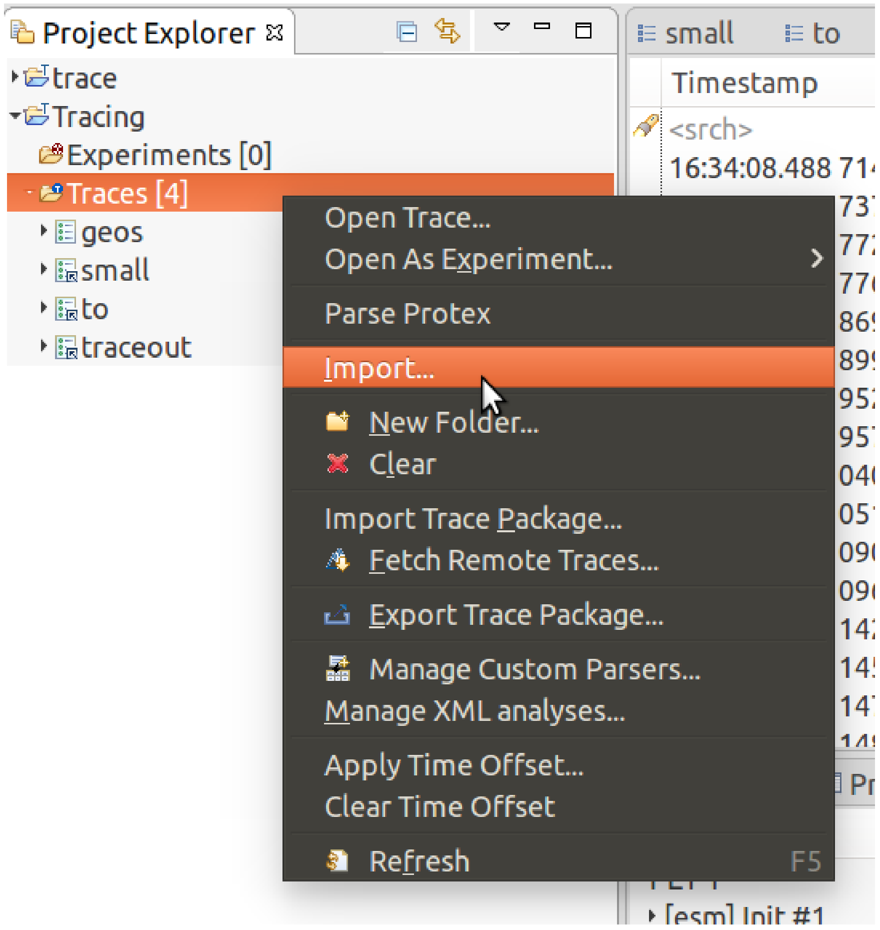
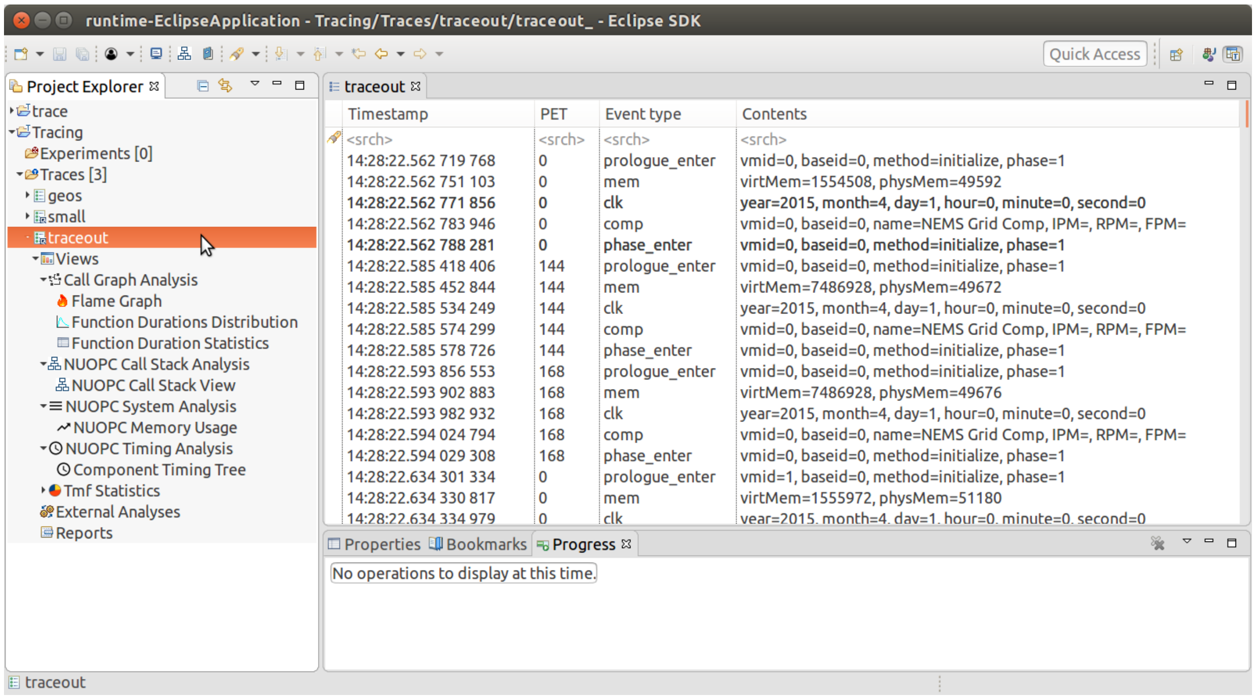


Fig. 3: Use the context menu to add traces to an existing project

3.2.2 Open the Trace

Double click on the trace in the Project Explorer to open the trace. You will see a table listing all of the events in the trace. Expand the “Views” element under the trace in the Project Explorer and you will see a list of available analyses and associated views.



Timestamp	PET	Event type	Contents
<srch>	<srch>	<srch>	<srch>
14:28:22.562 719 768	0	prologue_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.562 751 103	0	mem	virtMem=1554508, physMem=49592
14:28:22.562 771 856	0	clk	year=2015, month=4, day=1, hour=0, minute=0, second=0
14:28:22.562 783 946	0	comp	vmid=0, baseid=0, name=NEMS Grid Comp, IPM=, RPM=, FPM=
14:28:22.562 788 281	0	phase_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.585 418 406	144	prologue_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.585 452 844	144	mem	virtMem=7486928, physMem=49672
14:28:22.585 534 249	144	clk	year=2015, month=4, day=1, hour=0, minute=0, second=0
14:28:22.585 574 299	144	comp	vmid=0, baseid=0, name=NEMS Grid Comp, IPM=, RPM=, FPM=
14:28:22.585 578 726	144	phase_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.593 856 553	168	prologue_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.593 902 883	168	mem	virtMem=7486928, physMem=49676
14:28:22.593 982 932	168	clk	year=2015, month=4, day=1, hour=0, minute=0, second=0
14:28:22.594 024 794	168	comp	vmid=0, baseid=0, name=NEMS Grid Comp, IPM=, RPM=, FPM=
14:28:22.594 029 308	168	phase_enter	vmid=0, baseid=0, method=initialize, phase=1
14:28:22.634 301 334	0	prologue_enter	vmid=1, baseid=0, method=initialize, phase=1
14:28:22.634 330 817	0	mem	virtMem=1555972, physMem=51180
14:28:22.634 334 979	0	clk	year=2015, month=4, day=1, hour=0, minute=0, second=0

Fig. 4: The Project Explorer on the left shows all of the available analyses for the trace and associated views under each analysis. On the right is the raw list of events in the trace.

Important: If you do not see a set of analyses under the trace in the Project Explorer, but instead a list of files, you need to switch to the “Tracing” perspective. From the menu select **Window->Perspective->Open Perspective->Other. . .**, find the “Tracing” perspective and click open.

3.3 NUOPC Call Stack Analysis

The NUOPC Call Stack view shows visually the entry and exit points of each NUOPC/ESMF phase in the traced PETs. The PETs are aligned in time vertically so that it is easy to understand concurrency in the system. This view is helpful for seeing the hierarchical order of execution of component phases and for assessing load imbalance. The view is organized first by host/node (i.e., in a supercomputing environment) and then by PET number.

Open the NUOPC Call Stack View by double-clicking “NUOPC Call Stack View” in the Project Explorer under the imported trace. It is under Views / NUOPC Call Stack Analysis (see figure below).

For each PET, the view shows initialize, run, and finalize component execution phases and timing information about each phase.

The NUOPC Call Stack View toolbar allows you to navigate the view. If the trace is large (in terms of number of events or PETs), the call stack view may take a few seconds or longer to populate. Click the house icon to zoom out to the full execution trace.

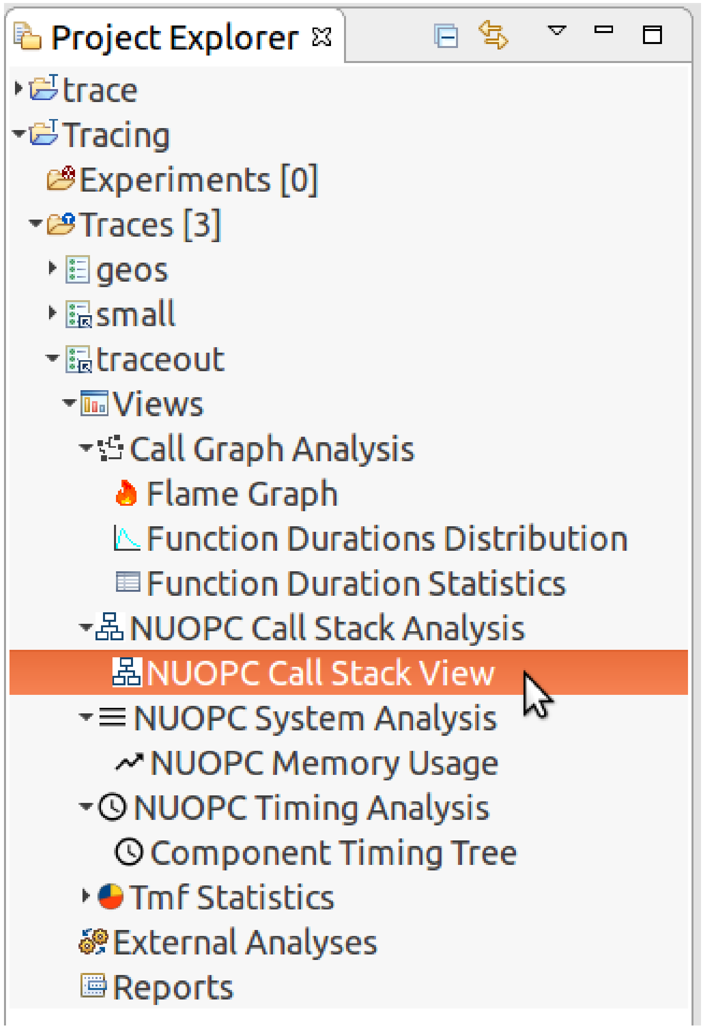


Fig. 5: Double-click "NUOPC Call Stack View" in the Project Explorer to open up the view.

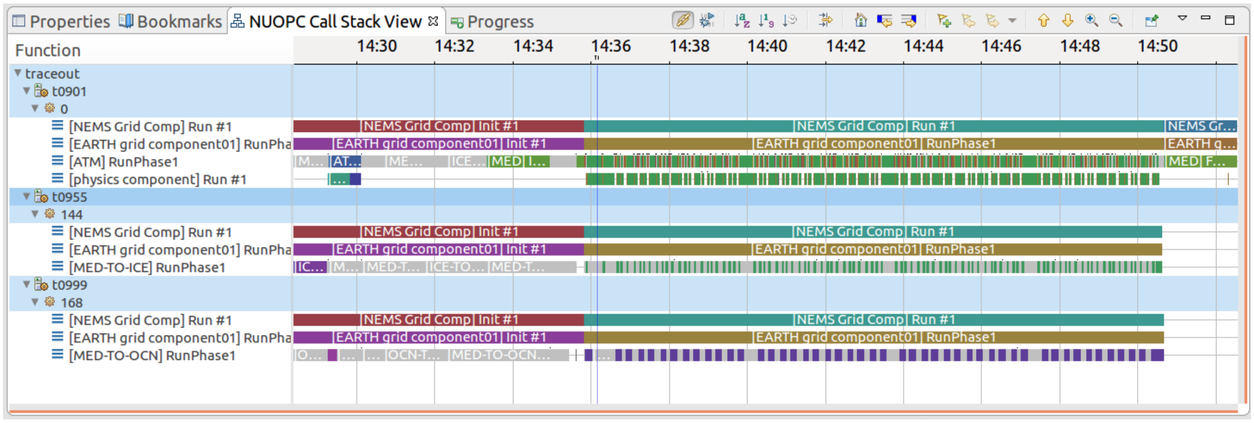


Fig. 6: The NUOPC Call Stack View showing three PETs

- The house icon zooms out to see the full execution trace.
- The + and - magnifying glass will zoom in and out.
- Right-click (CTRL-click on Mac), hold, and drag to zoom in on a particular time window.
- Left-click (CTRL-click on Max), hold, and drag to select a region and see the time delta at the bottom of the window.

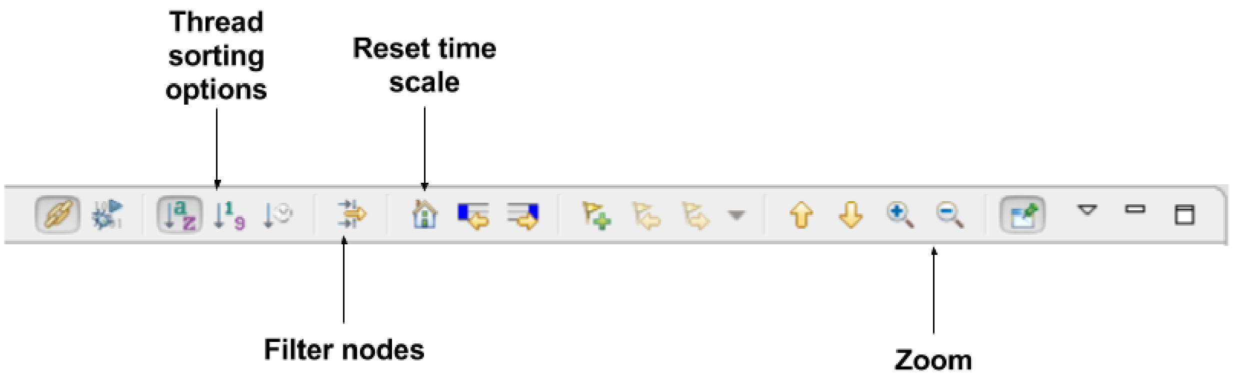
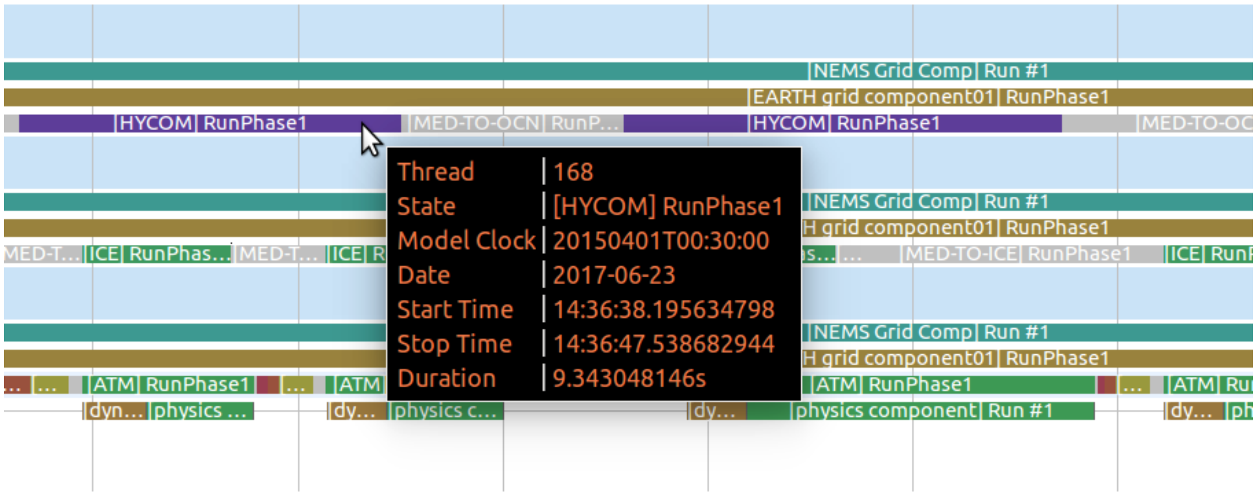


Fig. 7: The Call Stack toolbar.

Hover over a call stack element to see detailed timing information as well as the current model time on the ESMF clock in that phase.



```
Thread      | 168
State       | [HYCOM] RunPhase1
Model Clock | 20150401T00:30:00
Date        | 2017-06-23
Start Time  | 14:36:38.195634798
Stop Time   | 14:36:47.538682944
Duration    | 9.343048146s
```

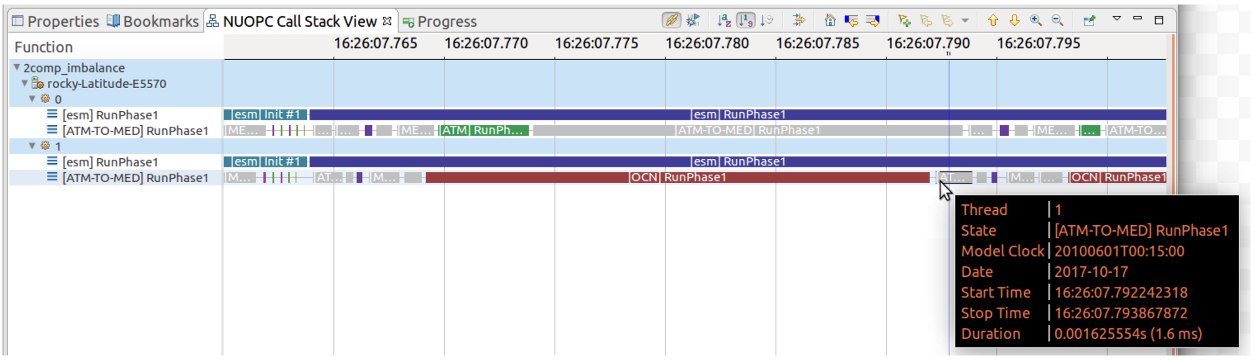
Fig. 8: Detailed timing information shown by hovering over a call stack element.

3.3.1 Check for Load Imbalance

In a coupled system with components running concurrently, ensuring a good load balance is important for computational efficiency. In NUOPC-based systems, concurrent components are assigned disjoint sets of PETs. In order to check for imbalance using the Call Stack View, make sure you trace a set of PETs that covers at least one PET of each component. A good approach is to trace the root (smallest) PET of each component. As stated above in the *Generate a Trace of a NUOPC Application* section, set the environment variable *ESMF_RUNTIME_TRACE_PETLIST* before executing the application to control which PETs to trace.

A clue that there is a load imbalance in the system is that too much time is spent inside NUOPC Connectors. Connectors are the primary communication components–they are responsible for moving data between Models and Mediators. If a system has a load imbalance, then unnecessary time will be spent inside Connectors when one component finishes its computation before another and must wait for data to be delivered by the Connector.

In the Call Stack View, Connectors are colored grey. The example trace shown below demonstrates a load imbalance. The first process, PET 0 executes the ATM component (shown in green), and the second process, PET 1, executes the OCN component (shown in red) concurrently. The ATM component finishes its RunPhase1 execution and enters the ATM-TO-MED Connector. The OCN component's RunPhase1 executes concurrently, but requires about four times as much execution time. When the OCN PET finishes its RunPhase1 it enters the ATM-TO-MED Connector as well, and both PETs are then able to proceed. The load imbalance means that PET 0 sits idle for a significant portion of time.



Thread	1
State	[ATM-TO-MED] RunPhase1
Model Clock	20100601T00:15:00
Date	2017-10-17
Start Time	16:26:07.792242318
Stop Time	16:26:07.793867872
Duration	0.001625554s (1.6 ms)

Fig. 9: A load imbalance in which the [ATM] RunPhase1 (shown in green) finishes before [OCN] RunPhase1 and wait idle inside the ATM-TO-MED Connector.

Load imbalance is possible whenever two or more components are running concurrently. One way to address this is to re-balance the PET counts so that more expensive components receive more PETs. The optimal PET count per component depends on a lot of factors, including the scalability of each component.

3.4 NUOPC Timing Analysis

3.4.1 Component Timing Tree

The Component Timing Tree shows timing statistics for NUOPC execution phases as well as user-defined regions in the trace. The top level elements in the timing tree are the PET numbers. (The timers are NOT aggregated across PETs.) Then, under each PET, the timing statistics are organized hierarchically to mirror the component tree structure of that PET. The tree can be sorted in ascending and descending order by each of the statistics by clicking on the column headings.

The statistics in the tree include:

Total time total aggregate time spent in the region, inclusive of all sub-regions

Self time total aggregate time spent in the region, excluding all sub-regions

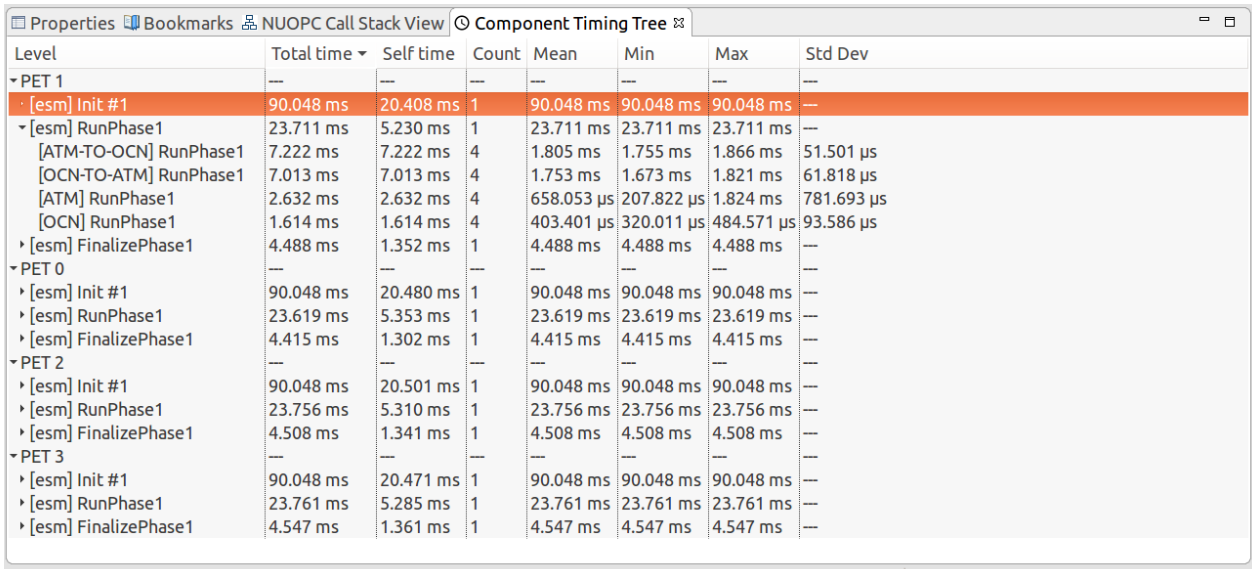
Count number of times the region is entered (called)

Mean average time per execution of the region, inclusive of all sub-regions

Min minimum execution time among calls into the region, inclusive of all sub-regions

Max maximum execution time among calls into the region, inclusive of all sub-regions

Std Dev standard deviation of execution times among calls into the region, inclusive of all sub-regions



Level	Total time	Self time	Count	Mean	Min	Max	Std Dev
PET 1	—	—	—	—	—	—	—
[esm] Init #1	90.048 ms	20.408 ms	1	90.048 ms	90.048 ms	90.048 ms	—
[esm] RunPhase1	23.711 ms	5.230 ms	1	23.711 ms	23.711 ms	23.711 ms	—
[ATM-TO-OCN] RunPhase1	7.222 ms	7.222 ms	4	1.805 ms	1.755 ms	1.866 ms	51.501 μs
[OCN-TO-ATM] RunPhase1	7.013 ms	7.013 ms	4	1.753 ms	1.673 ms	1.821 ms	61.818 μs
[ATM] RunPhase1	2.632 ms	2.632 ms	4	658.053 μs	207.822 μs	1.824 ms	781.693 μs
[OCN] RunPhase1	1.614 ms	1.614 ms	4	403.401 μs	320.011 μs	484.571 μs	93.586 μs
[esm] FinalizePhase1	4.488 ms	1.352 ms	1	4.488 ms	4.488 ms	4.488 ms	—
PET 0	—	—	—	—	—	—	—
[esm] Init #1	90.048 ms	20.480 ms	1	90.048 ms	90.048 ms	90.048 ms	—
[esm] RunPhase1	23.619 ms	5.353 ms	1	23.619 ms	23.619 ms	23.619 ms	—
[esm] FinalizePhase1	4.415 ms	1.302 ms	1	4.415 ms	4.415 ms	4.415 ms	—
PET 2	—	—	—	—	—	—	—
[esm] Init #1	90.048 ms	20.501 ms	1	90.048 ms	90.048 ms	90.048 ms	—
[esm] RunPhase1	23.756 ms	5.310 ms	1	23.756 ms	23.756 ms	23.756 ms	—
[esm] FinalizePhase1	4.508 ms	1.341 ms	1	4.508 ms	4.508 ms	4.508 ms	—
PET 3	—	—	—	—	—	—	—
[esm] Init #1	90.048 ms	20.471 ms	1	90.048 ms	90.048 ms	90.048 ms	—
[esm] RunPhase1	23.761 ms	5.285 ms	1	23.761 ms	23.761 ms	23.761 ms	—
[esm] FinalizePhase1	4.547 ms	1.361 ms	1	4.547 ms	4.547 ms	4.547 ms	—

Fig. 10: The Component Timing Tree view is organized according to the component hierarchy.

Keep in mind that regions can appear at multiple places in the hierarchy. The statistics in the tree are relevant for that particular location in the hierarchy. For example, the "Total time" spent in a region means the aggregate time of the all calls to the region *at that place in the hierarchy*.

3.4.2 Timing User-defined Regions

Timing user-defined regions is supported by inserting calls to *ESMF_TraceRegionEnter()* and *ESMF_TraceRegionExit()* into the application code and generating a trace. See the tracing section of the ESMF reference manual for more information.

User-defined regions will appear in the Component Timing Tree at their proper nesting level.

3.4.3 Flame Graph

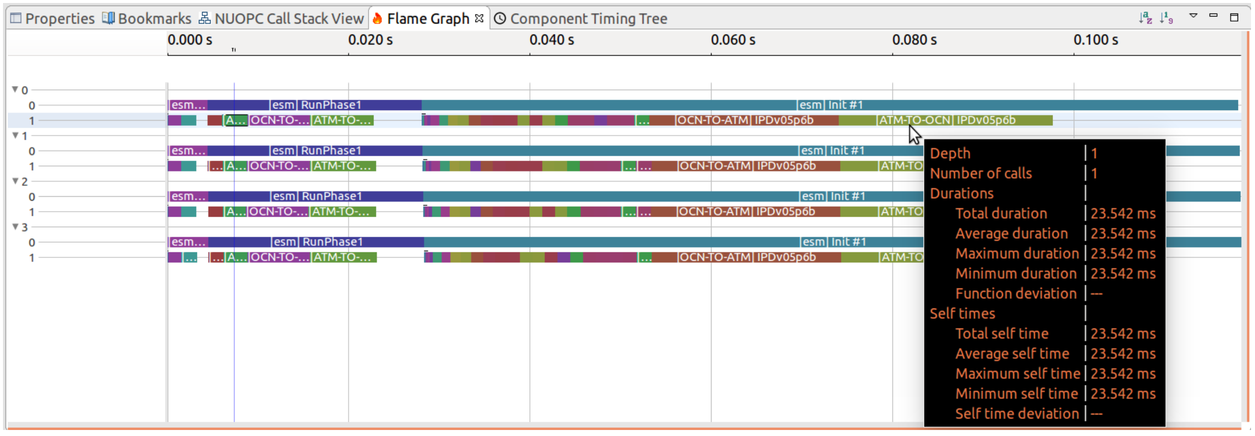
The Flame Graph shows the same statistics available in the Component Timing Tree in a visual form. The Flame Graph is an aggregated form of the Call Stack View, organized by depth and then region at that depth. This allows you to quickly see where most of the time is spent in the application when deciding where to optimize.

The Flame Graph is provided by the TraceCompass plugin, and more detailed information about this view is available in the TraceCompass user guide.

3.4.4 Function Duration Statistics

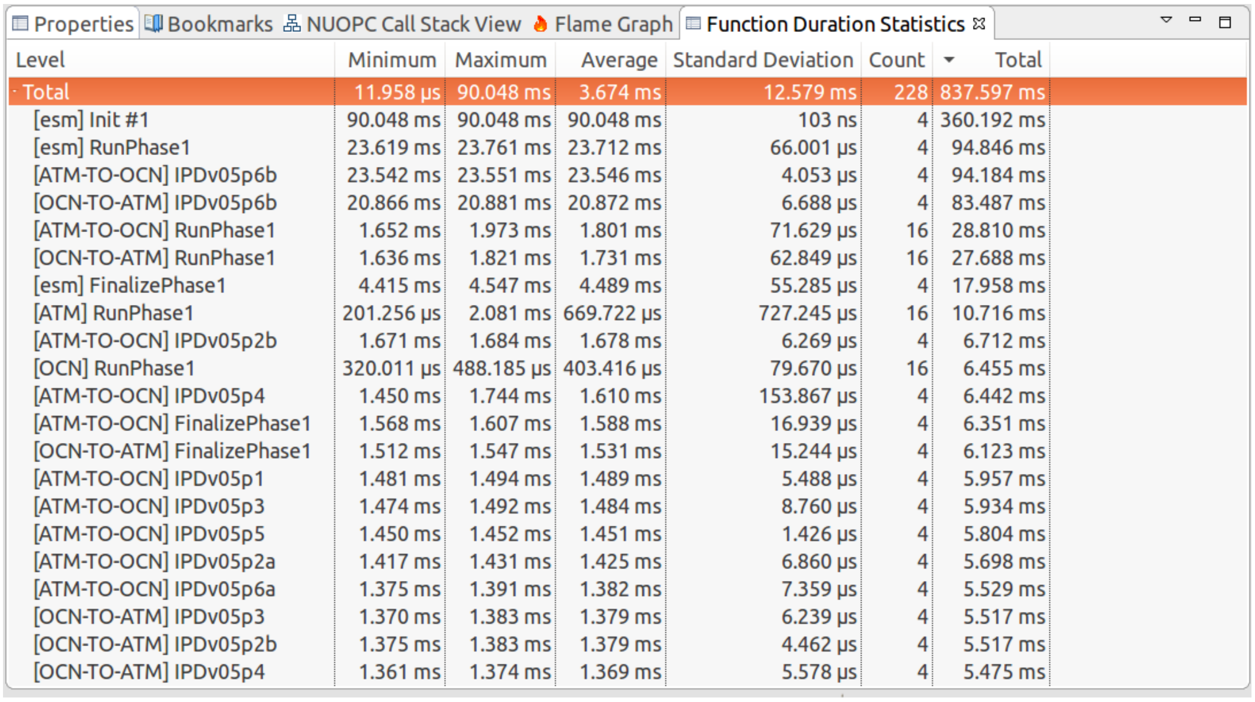
The Function Duration Statistics view is a flat list of all the regions, including component execution phases and user-defined regions. Unlike the Component Timing Tree, these statistics are aggregated across all PETs in the trace.

The Function Duration Statistics View is provided by the TraceCompass plugin, and more detailed information about this view is available in the TraceCompass user guide.



Depth	1
Number of calls	1
Durations	
Total duration	23.542 ms
Average duration	23.542 ms
Maximum duration	23.542 ms
Minimum duration	23.542 ms
Function deviation	---
Self times	
Total self time	23.542 ms
Average self time	23.542 ms
Maximum self time	23.542 ms
Minimum self time	23.542 ms
Self time deviation	---

Fig. 11: The Flame Graph view



Level	Minimum	Maximum	Average	Standard Deviation	Count	Total
Total	11.958 µs	90.048 ms	3.674 ms	12.579 ms	228	837.597 ms
[esm] Init #1	90.048 ms	90.048 ms	90.048 ms	103 ns	4	360.192 ms
[esm] RunPhase1	23.619 ms	23.761 ms	23.712 ms	66.001 µs	4	94.846 ms
[ATM-TO-OCN] IPDv05p6b	23.542 ms	23.551 ms	23.546 ms	4.053 µs	4	94.184 ms
[OCN-TO-ATM] IPDv05p6b	20.866 ms	20.881 ms	20.872 ms	6.688 µs	4	83.487 ms
[ATM-TO-OCN] RunPhase1	1.652 ms	1.973 ms	1.801 ms	71.629 µs	16	28.810 ms
[OCN-TO-ATM] RunPhase1	1.636 ms	1.821 ms	1.731 ms	62.849 µs	16	27.688 ms
[esm] FinalizePhase1	4.415 ms	4.547 ms	4.489 ms	55.285 µs	4	17.958 ms
[ATM] RunPhase1	201.256 µs	2.081 ms	669.722 µs	727.245 µs	16	10.716 ms
[ATM-TO-OCN] IPDv05p2b	1.671 ms	1.684 ms	1.678 ms	6.269 µs	4	6.712 ms
[OCN] RunPhase1	320.011 µs	488.185 µs	403.416 µs	79.670 µs	16	6.455 ms
[ATM-TO-OCN] IPDv05p4	1.450 ms	1.744 ms	1.610 ms	153.867 µs	4	6.442 ms
[ATM-TO-OCN] FinalizePhase1	1.568 ms	1.607 ms	1.588 ms	16.939 µs	4	6.351 ms
[OCN-TO-ATM] FinalizePhase1	1.512 ms	1.547 ms	1.531 ms	15.244 µs	4	6.123 ms
[ATM-TO-OCN] IPDv05p1	1.481 ms	1.494 ms	1.489 ms	5.488 µs	4	5.957 ms
[ATM-TO-OCN] IPDv05p3	1.474 ms	1.492 ms	1.484 ms	8.760 µs	4	5.934 ms
[ATM-TO-OCN] IPDv05p5	1.450 ms	1.452 ms	1.451 ms	1.426 µs	4	5.804 ms
[ATM-TO-OCN] IPDv05p2a	1.417 ms	1.431 ms	1.425 ms	6.860 µs	4	5.698 ms
[ATM-TO-OCN] IPDv05p6a	1.375 ms	1.391 ms	1.382 ms	7.359 µs	4	5.529 ms
[OCN-TO-ATM] IPDv05p3	1.370 ms	1.383 ms	1.379 ms	6.239 µs	4	5.517 ms
[OCN-TO-ATM] IPDv05p2b	1.375 ms	1.383 ms	1.379 ms	4.462 µs	4	5.517 ms
[OCN-TO-ATM] IPDv05p4	1.361 ms	1.374 ms	1.369 ms	5.578 µs	4	5.475 ms

Fig. 12: The Function Duration Statistics view aggregates each region across all PETs in the trace.

CHAPTER 4

Working with NUOPC Code

4.1 Create a Fortran Project with Your Model Code

There are two options for creating a Fortran project in Eclipse based on whether the source code you are importing is local or on a remote machine. The simplest approach is to have the source code available locally. However, that is not always practical so Eclipse provides a synchronization capability with files on a remote system accessible via SSH. The sections below describe briefly how to create these two kinds of projects.

See also:

This user guide provides only high level guidance in setting up local and remote Fortran projects. More details can be found on the Parallel Tools Platform documentation site.

4.1.1 Projects with Local Files

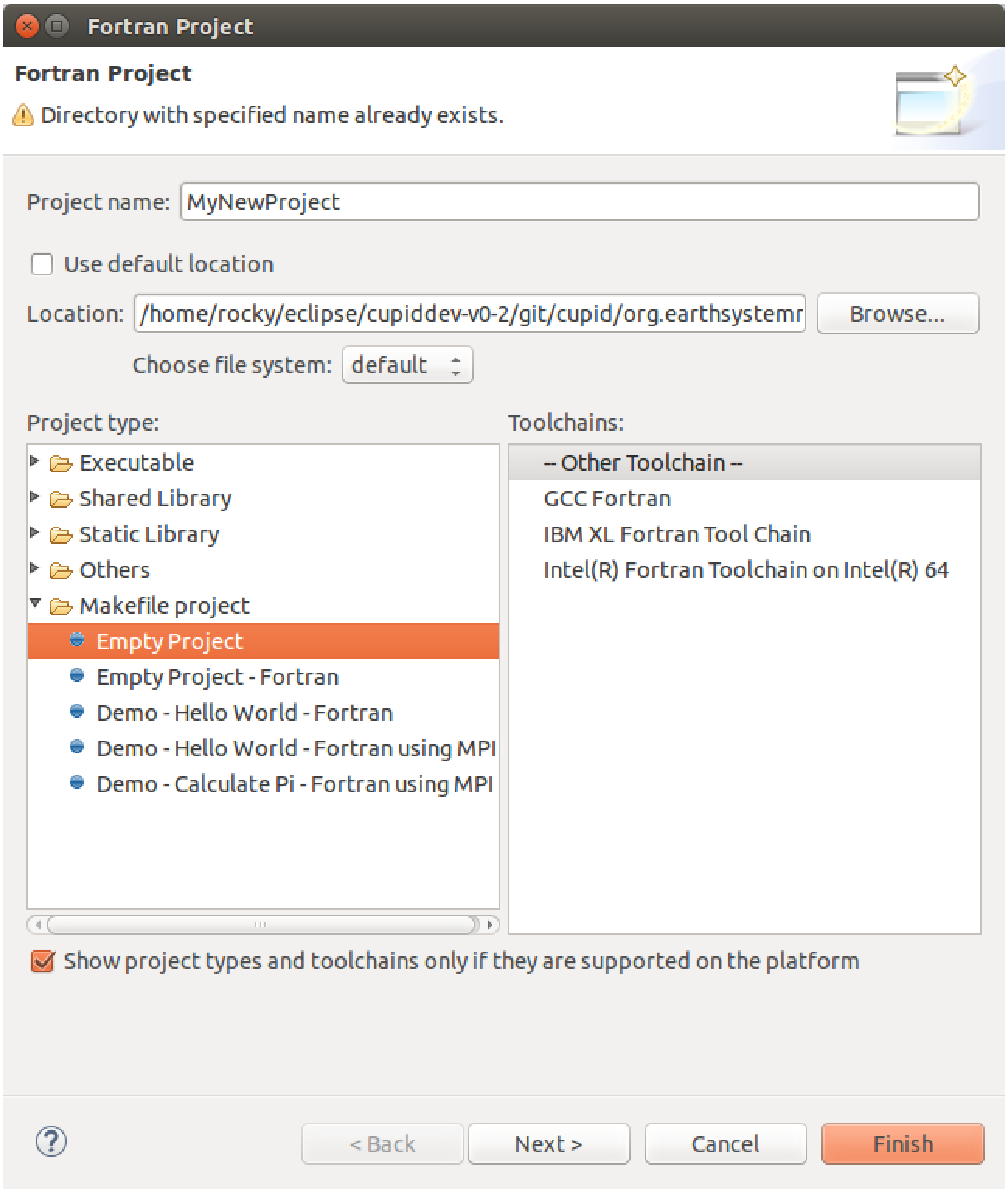
To create a new Fortran project with local files, right-click (CTRL-click on Mac) on the Project Explorer and select **New -> Fortran Project**. On the New Project screen you can un-check *Use default location* and browse to the location of the files. If you use the default location, the project folder will be in the Eclipse workspace folder and you will need to import files manually by selecting **File -> Import…** from the menu after creating the project.

Under *Project type*, it is recommended that you select Empty Project under the Makefile project folder. (The project will not actually be empty if you selected the location of your local files.) Click Finish and the new project will be created and will appear in the Project Explorer.

4.1.2 Synchronized Projects with Remote Files

A synchronized Fortran project will copy files from a remote file system and ensure that the remote and local copies stay synchronized. This is convenient if the code will be built and executed on a remote system. The disadvantage of this approach is that the initial synchronization can take multiple minutes if the size of the source tree is large. However, once the initial synchronization is complete, only changed files need be communicated over the network.

The first step is set up the connection with the remote machine. Open the Connections view by selecting **Window -> Show View -> Other**. In the list of views, filter for “Connections” and click OK to show the view.



Fortran Project

Fortran Project

⚠ Directory with specified name already exists.

Project name: MyNewProject

☐ Use default location

Location: /home/rocky/eclipse/cupiddev-v0-2/git/cupid/org.earthsystemr

Choose file system: default

Project type:

- Executable
- Shared Library
- Static Library
- Others
- Makefile project
  - Empty Project
  - Empty Project - Fortran
  - Demo - Hello World - Fortran
  - Demo - Hello World - Fortran using MPI
  - Demo - Calculate Pi - Fortran using MPI

Toolchains:

- -- Other Toolchain --
- GCC Fortran
- IBM XL Fortran Tool Chain
- Intel(R) Fortran Toolchain on Intel(R) 64

☑ Show project types and toolchains only if they are supported on the platform

Fig. 1: The New Fortran Project wizard.

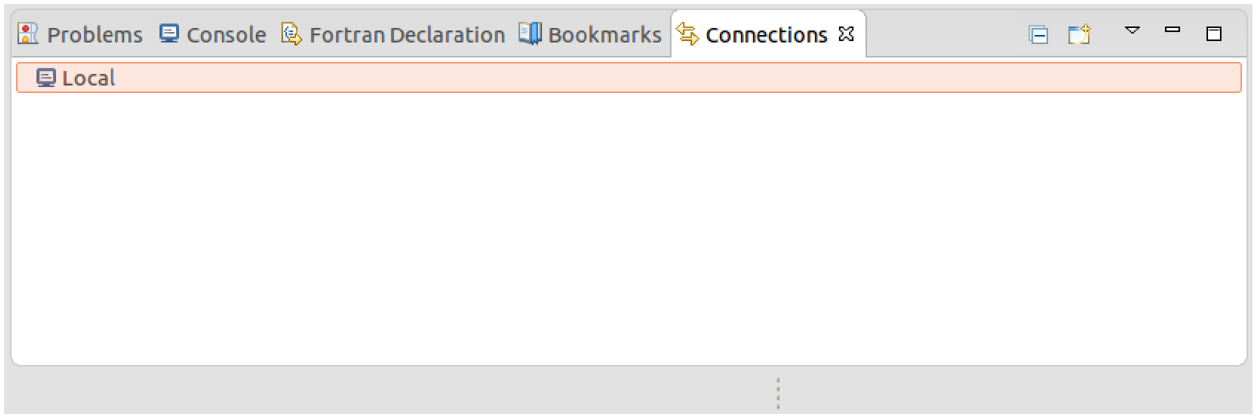


Fig. 2: The Connections view.

Create a new connection by clicking the New Connection button (with a small yellow +) in the toolbar on the Connections view. Choose SSH connection on the following screen and click next. On the next screen fill in the details about the connection. The password can be left blank and you will be prompted at each login. In some cases you may need to create multiple connections and use one as the proxy for another, for example, if you must first authenticate through a login node. Click Finish when you are done and you will see the new connection in Connetions view.

Now create a new synchronized Fortran project right-click in the Project Explorer and select **New -> Synchronized Fortran Project**. Fill in the project name, select the remote connection you created and fill in the file path to the root of the source code on the remote system.

You can optionally filter which files are synchronized by clicking **Modify file filtering…** and choosing certain directories to exclude. In particular, directories containing large data files and other non-source code should be excluded to speed up the synchronization.

Under *Project Type* select Empty Project under Makefile project. Selecting local and remote toolchains is not required unless you plan to use the Eclipse build system. Click Finish and the new project will appear in the Project Explorer.

The project will initially be empty and you will need to manually kick off the first synchronization. Do this by clicking the synchronize button in the toolbar or by right-clicking (CTRL-click on mac) the project folder and selecting **Synchronize -> Sync Active Now**. Remote files will be copied to the local workspace. By default, future synchronizations will happen automatically when changes are made to local files. If the remote files change, or if you notice that changes have not been propagated to the remote system, force a sync using the procedure above.

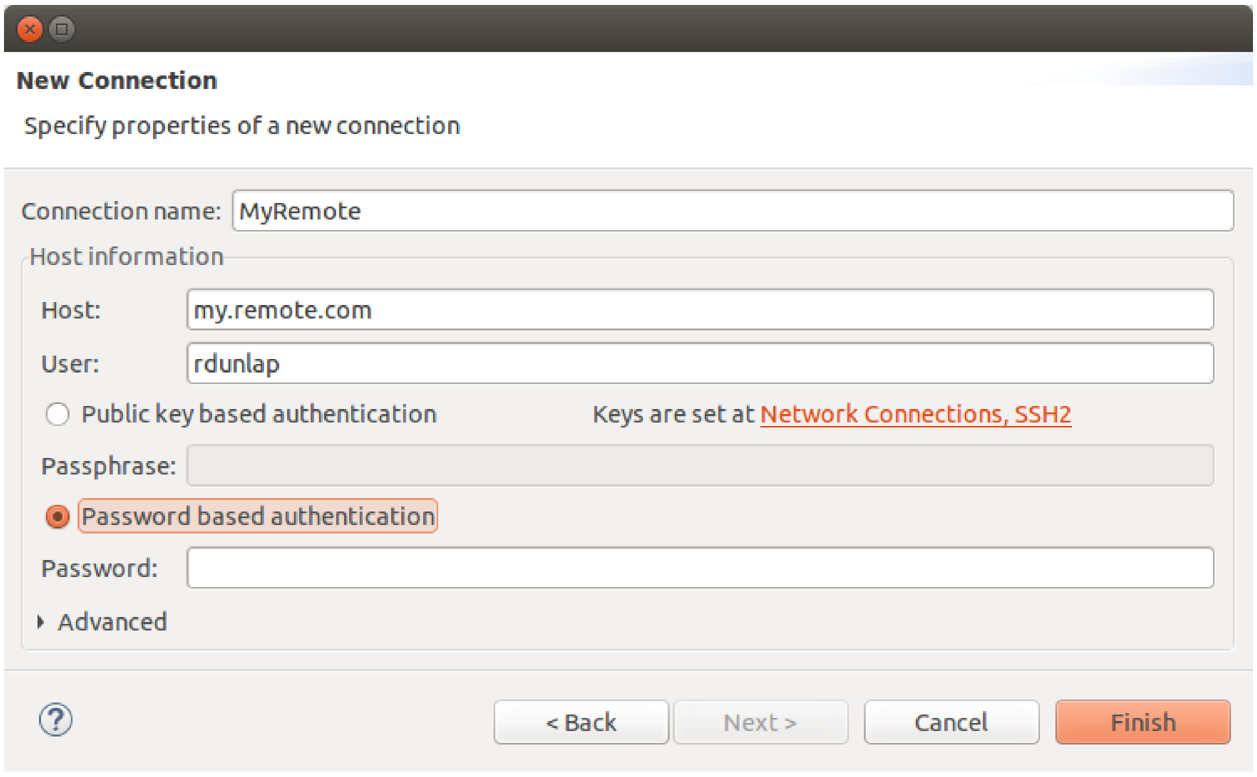
4.1.3 Ensure Fortran Analysis is Enabled

Important: Turning on the Fortran analysis/refactoring engine is required for Cupid to work properly.

Cupid depends on the Fortran analysis engine being activated for projects containing NUOPC code. By default it is turned off. To turn it on for a project, right-click (CTRL-click on Mac) on the project folder and select **Properties**. Under **Fortran General -> Analysis/Refactoring** check the first box, *Enable Fortran analysis/refactoring*.

4.2 Reverse Engineer a NUOPC Cap

Cupid's reverse engineering function is capable of analyzing the source code of a NUOPC component to create a representation at a higher level of abstraction. The reverse engineering analysis is limited to only the NUOPC cap of


New Connection

Specify properties of a new connection

Connection name: MyRemote

Host information

Host: my.remote.com

User: rdunlap

Public key based authentication — Keys are set at Network Connections, SSH2

Passphrase:

[✓ Password based authentication]

Password:

Fig. 3: The New Connection wizard.

a component, which is typically a single Fortran module. The analysis does not descend into the model code itself. Once the higher level representation is obtained, Cupid is able to provide NUOPC-aware capabilities, such as basic validation of correct API usage and in-place code generation–i.e., weaving new code into the correct places of an existing source file. The reverse engineering analysis phase happens automatically as a background process and an index of NUOPC components in the workspace is maintained.

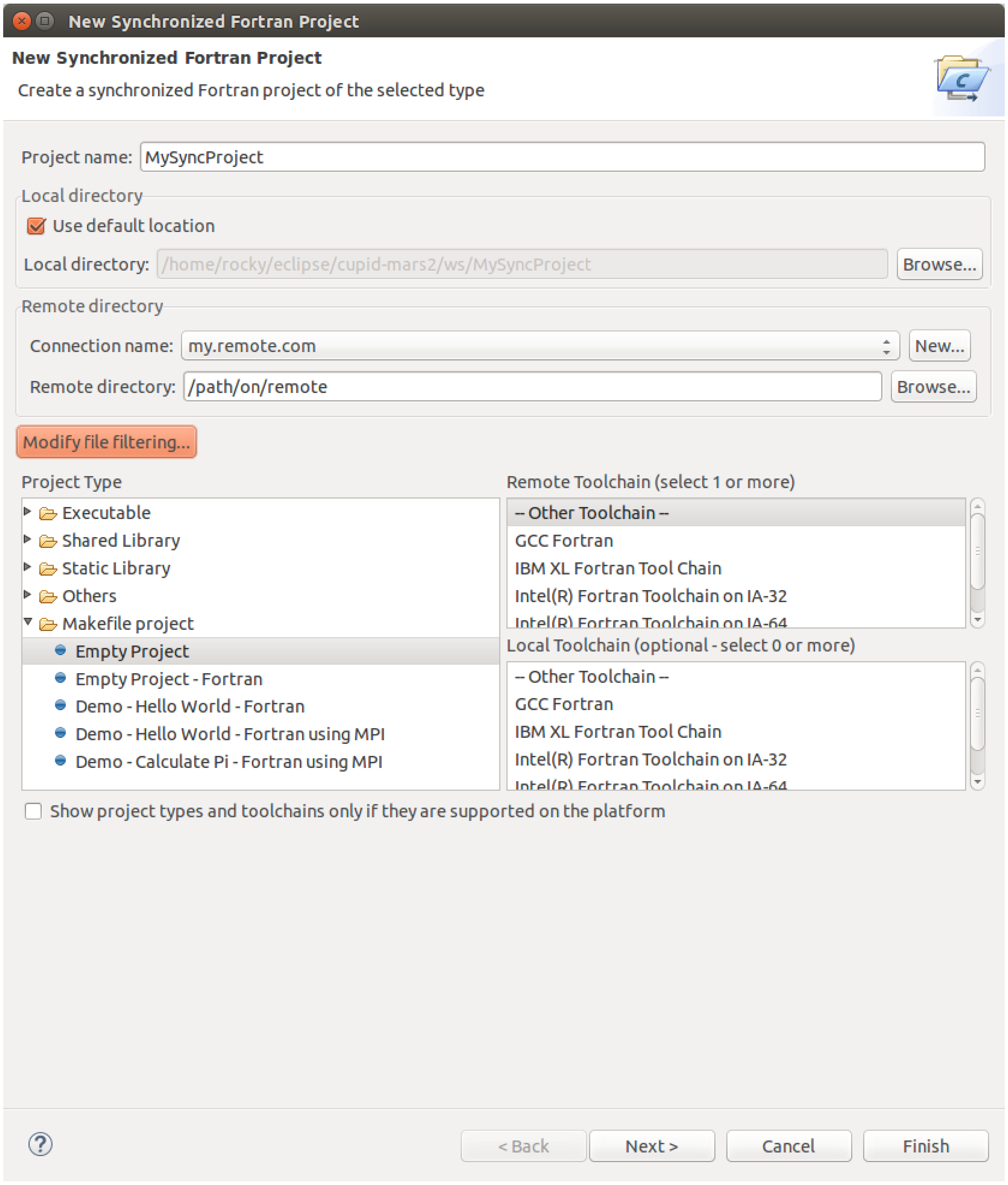
4.2.1 Show the NUOPC View

The results of the reverse engineered code can be seen in outline form in the NUOPC View.

The NUOPC View is set up to show whenever the Fortran perspective is selected. The current perspective is shown in the upper right-hand corner of Eclipse. There is also an Open Perspective button which can be used to select the Fortran perspective if it is not already shown.

There are other ways to show the NUOPC View:

- If the NUOPC View is not visible and you open a file with NUOPC code, a dialog will ask you if you would like to open the NUOPC View. This behavior can be turned off in the Cupid preferences (select **Window -> Preferences** from the menu and select Cupid in the list on the left).
- The main toolbar contains a Show NUOPC View button, circled in green below


New Synchronized Fortran Project

New Synchronized Fortran Project

Create a synchronized Fortran project of the selected type

Project name: MySyncProject

Local directory

[✓] Use default location

Local directory: /home/rocky/eclipse/cupid-mars2/ws/MySyncProject

Remote directory

Connection name: my.remote.com

Remote directory: /path/on/remote

Project Type

- Executable
- Shared Library
- Static Library
- Others
- Makefile project
  - Empty Project
  - Empty Project - Fortran
  - Demo - Hello World - Fortran
  - Demo - Hello World - Fortran using MPI
  - Demo - Calculate Pi - Fortran using MPI

Remote Toolchain (select 1 or more)

- -- Other Toolchain --
- GCC Fortran
- IBM XL Fortran Tool Chain
- Intel(R) Fortran Toolchain on IA-32
- Intel(R) Fortran Toolchain on IA-64

Local Toolchain (optional - select 0 or more)

- -- Other Toolchain --
- GCC Fortran
- IBM XL Fortran Tool Chain
- Intel(R) Fortran Toolchain on IA-32
- Intel(R) Fortran Toolchain on IA-64

[ ] Show project types and toolchains only if they are supported on the platform

Fig. 4: The New Synchronized Fortran Project wizard.

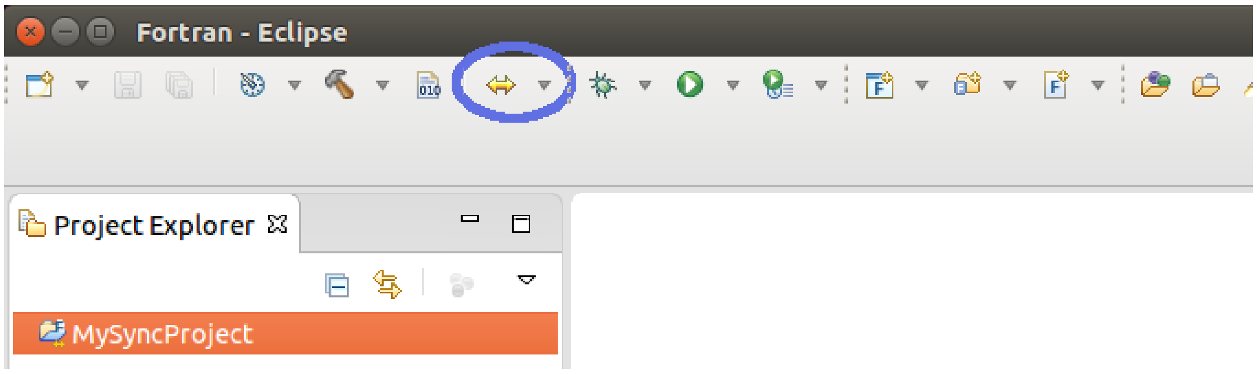
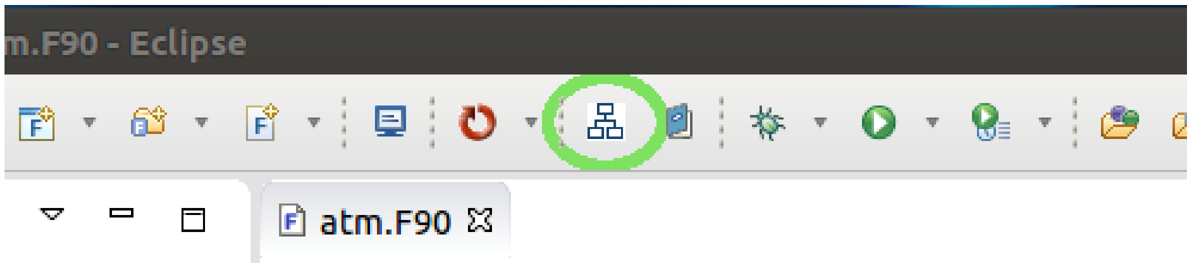


Fig. 5: After selecting a project, click the Synchronize button on the toolbar (circled in blue) to kick off the first synchronization. Remote files will be copied to the local workspace.



- The NUOPC View can be accessed from the **Window -> Show View -> Other** menu

The NUOPC View will automatically refresh itself as files are changed and saved in the workspace. It is also possible to force a refresh of the NUOPC View using the refresh button (blue circular arrow) in the top right corner of the NUOPC View. This will first ensure that the Fortran analysis database is up to date and then it will rebuild the index of NUOPC components in the workspace.

4.2.2 Elements in the NUOPC View outline

The top-level element in the NUOPC View tree are files in the workspace that contain code for a NUOPC component. The first element under each file indicates that type of component (Model, Driver, or Mediator). Sub-elements underneath the component type represent something in the source code, such as a SetServices subroutine, a NUOPC initialization subroutine, a specialization point subroutine, imports of NUOPC generic modules, or calls into the NUOPC API. Many of the elements have small icons: a blue circle with an M maps to a Fortran module, a green circle maps to subroutine, and a yellow arrow pointing to the right represents a subroutine or function call. If a green circle has a small upward triangle in the corner, it indicates that the subroutine is not in the current module, but is inherited from a NUOPC generic component. Grayed out items do not map to any source code element, but represent subroutines or API calls that can be generated. Red items indicate that there is a validation problem rooted at that element. Some elements indicate a cardinality such as [1..n], which indicates that one or more elements of that type can exist, or [0..1], which indicates the element is optional.

The outline is divided into several major sections:

- module imports (only specific ones are shown)
- SetServices
- initialization phases and specialization points
- run phases and specialization points
- finalize phases and specialization points

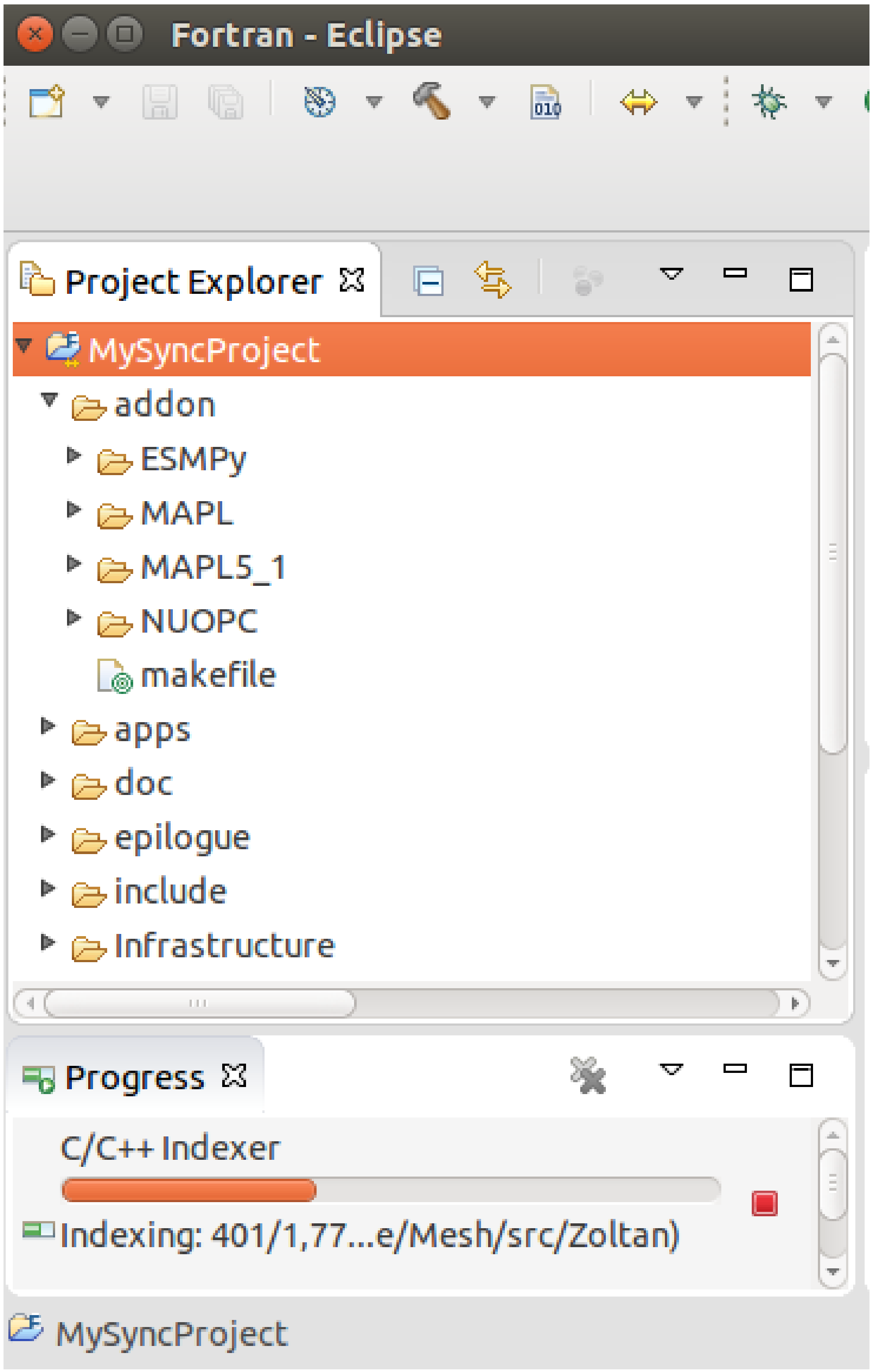
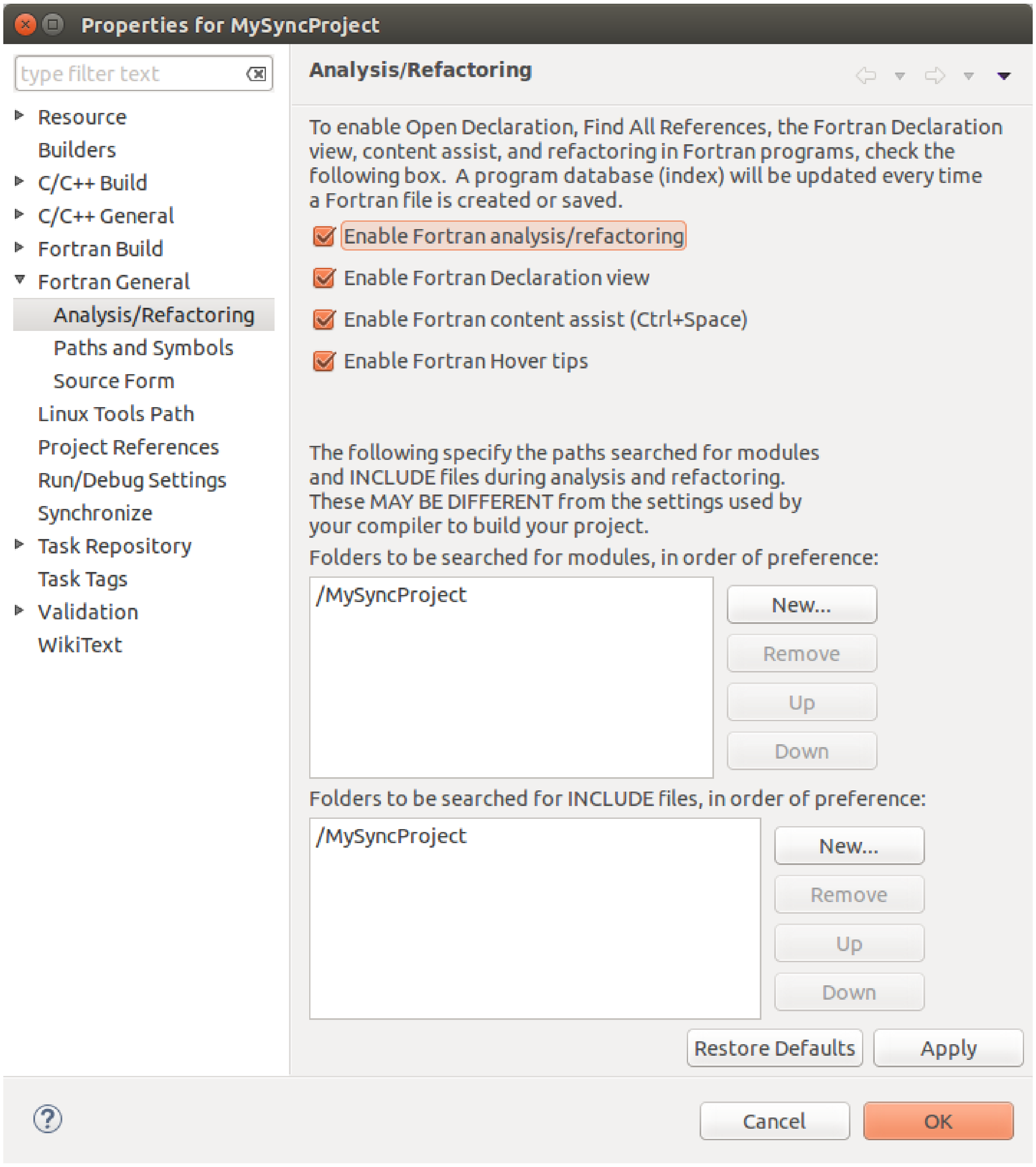


Fig. 6: After the synchronization process, files will be visible in the Project Explorer.



Analysis/Refactoring

To enable Open Declaration, Find All References, the Fortran Declaration view, content assist, and refactoring in Fortran programs, check the following box. A program database (index) will be updated every time a Fortran file is created or saved.

- [✓] Enable Fortran analysis/refactoring
- [✓] Enable Fortran Declaration view
- [✓] Enable Fortran content assist (Ctrl+Space)
- [✓] Enable Fortran Hover tips

The following specify the paths searched for modules and INCLUDE files during analysis and refactoring. These MAY BE DIFFERENT from the settings used by your compiler to build your project.

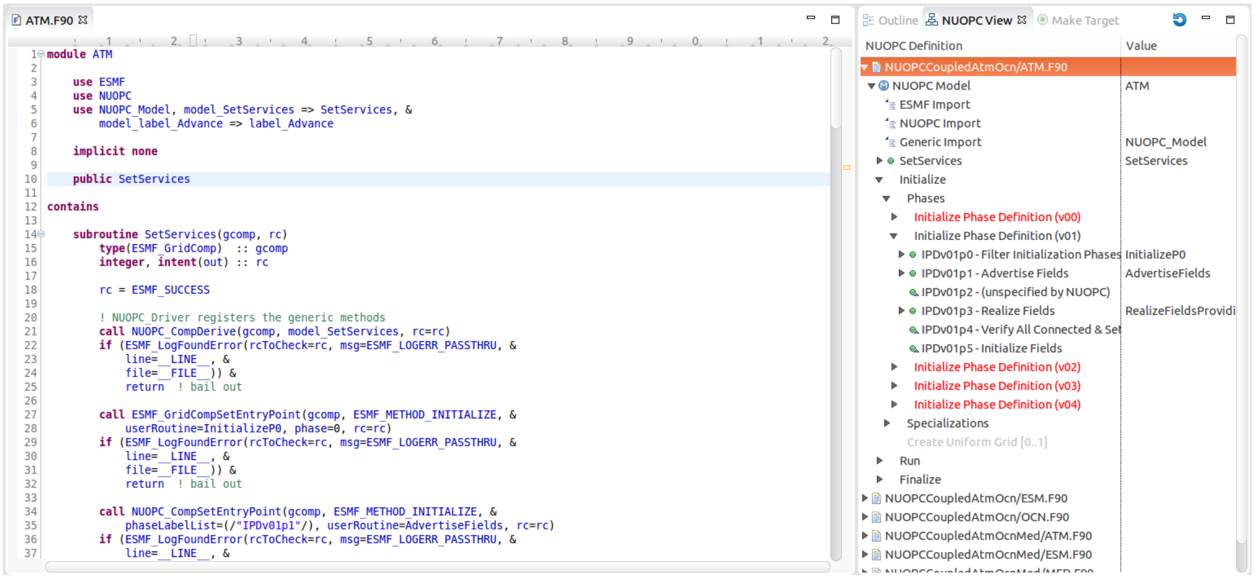
Folders to be searched for modules, in order of preference:

/MySyncProject

Folders to be searched for INCLUDE files, in order of preference:

/MySyncProject

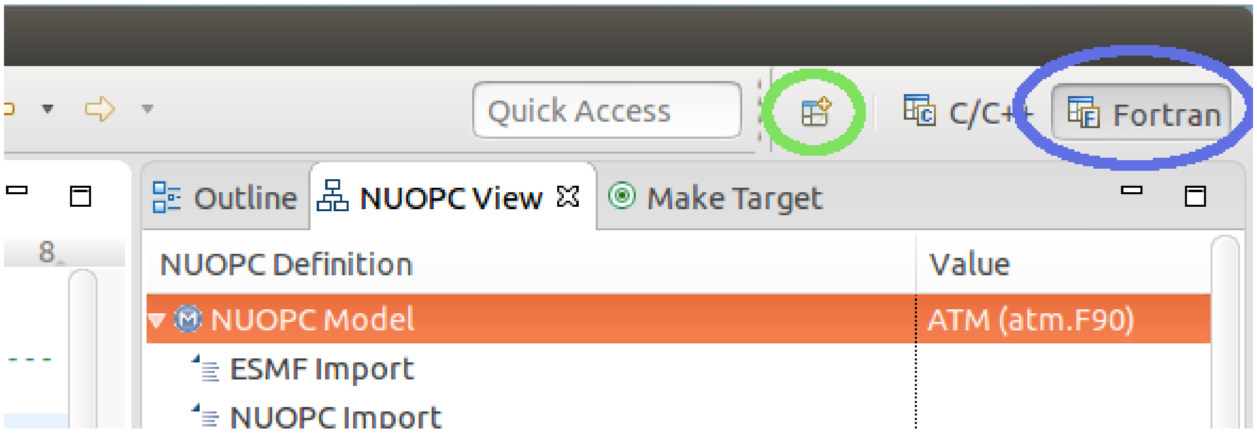
Fig. 7: Enable Fortran analysis/refactoring on in the project properties.



```fortran
module ATM

    use ESMF
    use NUOPC
    use NUOPC_Model, model_SetServices => SetServices, &
        model_label_Advance => label_Advance

    implicit none

    public SetServices

contains

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

        rc = ESMF_SUCCESS

        ! NUOPC_Driver registers the generic methods
        call NUOPC_CompDerive(gcomp, model_SetServices, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
            line=__LINE__, &
            file=__FILE__)) &
            return  ! bail out

        call ESMF_GridCompSetEntryPoint(gcomp, ESMF_METHOD_INITIALIZE, &
            userRoutine=InitializeP0, phase=0, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
            line=__LINE__, &
            file=__FILE__)) &
            return  ! bail out

        call NUOPC_CompSetEntryPoint(gcomp, ESMF_METHOD_INITIALIZE, &
            phaseLabelList=(/"IPDv01p1"/), userRoutine=AdvertiseFields, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
            line=__LINE__, &
```

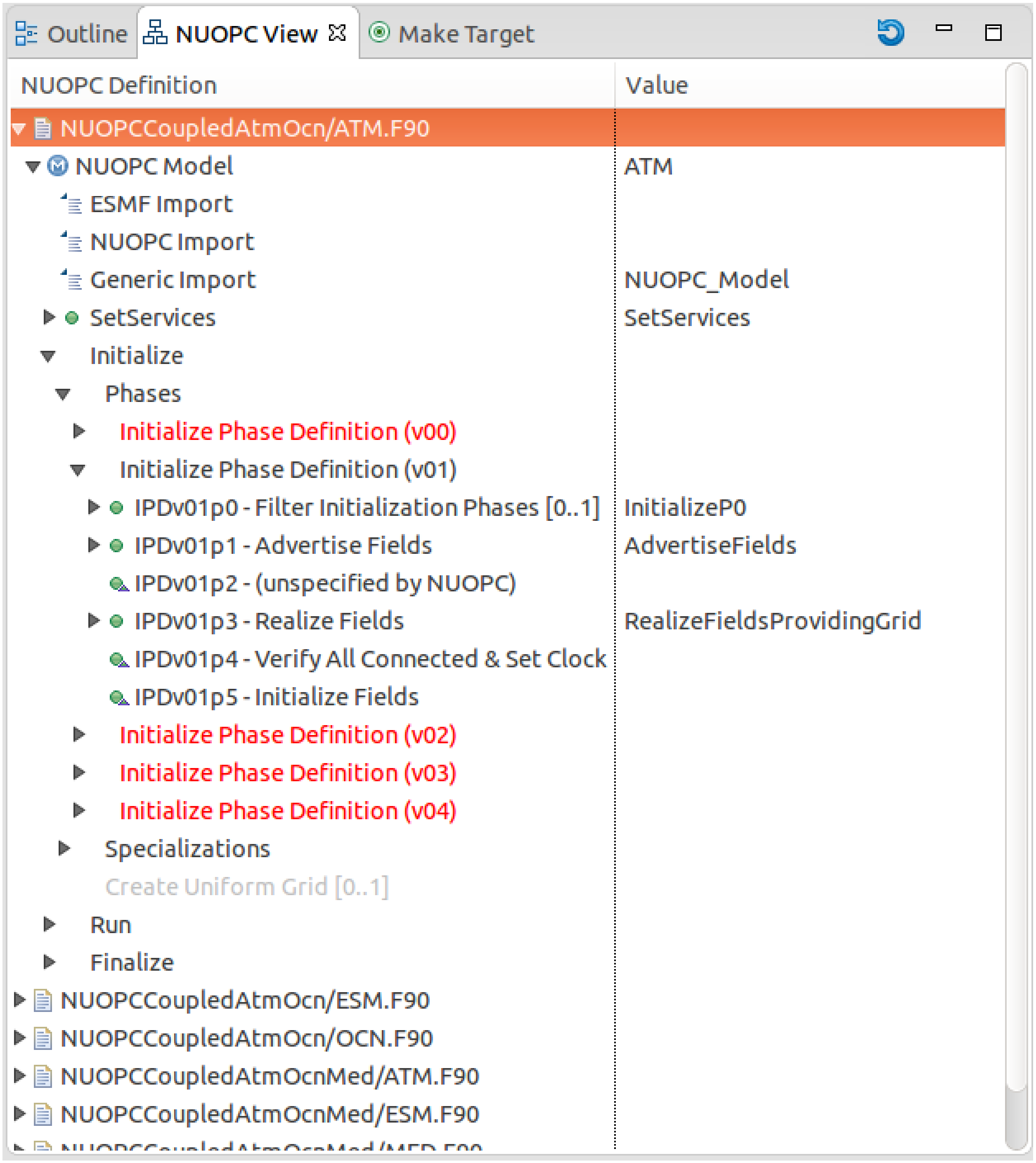
NUOPC Definition	Value
NUOPCCoupledAtmOcn/ATM.F90	
NUOPC Model	ATM
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
Initialize	
Phases	
Initialize Phase Definition (v00)	
Initialize Phase Definition (v01)	
IPDv01p0 - Filter Initialization Phases	InitializeP0
IPDv01p1 - Advertise Fields	AdvertiseFields
IPDv01p2 - (unspecified by NUOPC)	
IPDv01p3 - Realize Fields	RealizeFieldsProvidi
IPDv01p4 - Verify All Connected & Set	
IPDv01p5 - Initialize Fields	
Initialize Phase Definition (v02)	
Initialize Phase Definition (v03)	
Initialize Phase Definition (v04)	
Specializations	
Create Uniform Grid [0..1]	
Run	
Finalize	
NUOPCCoupledAtmOcn/ESM.F90	
NUOPCCoupledAtmOcn/OCN.F90	
NUOPCCoupledAtmOcnMed/ATM.F90	
NUOPCCoupledAtmOcnMed/ESM.F90	

Fig. 8: The NUOPC View (to the right of the source code) shows an outline of a reverse engineered NUOPC component.



NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	

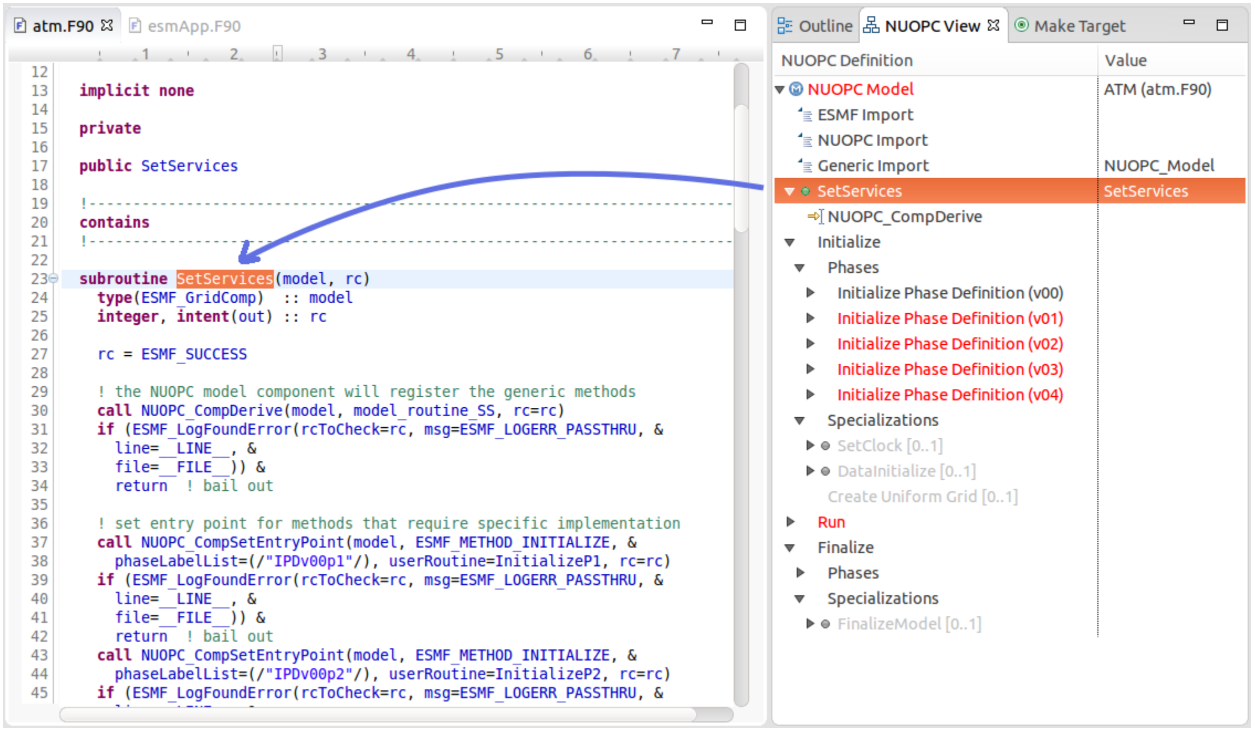
Fig. 9: The NUOPC View is set to appear automatically from the Fortran perspective (circled in blue). Click the Open Perspective button (circled in green) to open a new perspective.



NUOPC Definition	Value
NUOPCCoupledAtmOcn/ATM.F90	
NUOPC Model	ATM
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
Initialize	
Phases	
Initialize Phase Definition (v00)	
Initialize Phase Definition (v01)	
IPDv01p0 - Filter Initialization Phases [0..1]	InitializeP0
IPDv01p1 - Advertise Fields	AdvertiseFields
IPDv01p2 - (unspecified by NUOPC)	
IPDv01p3 - Realize Fields	RealizeFieldsProvidingGrid
IPDv01p4 - Verify All Connected & Set Clock	
IPDv01p5 - Initialize Fields	
Initialize Phase Definition (v02)	
Initialize Phase Definition (v03)	
Initialize Phase Definition (v04)	
Specializations	
Create Uniform Grid [0..1]	
Run	
Finalize	
NUOPCCoupledAtmOcn/ESM.F90	
NUOPCCoupledAtmOcn/OCN.F90	
NUOPCCoupledAtmOcnMed/ATM.F90	
NUOPCCoupledAtmOcnMed/ESM.F90	

Fig. 10: The NUOPC View showing an outline of a NUOPC Model cap.

The NUOPC View is linked to the source code in the active editor. To navigate to the source code related to the element, double-click the element. The relevant code segment will be brought into focus. If the element maps to a subroutine definition, the name of the subroutine will be highlighted. If the element maps to an API call, the call will be highlighted. If an element represents an inherited subroutine (a green circle with small triangle), then it does not appear in the current file, so no code will be highlighted when double-clicking the element.



```
implicit none

private

public SetServices

!-----------------------------------------------------------------------------
contains
!-----------------------------------------------------------------------------

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

  rc = ESMF_SUCCESS

  ! the NUOPC model component will register the generic methods
  call NUOPC_CompDerive(model, model_routine_SS, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=__LINE__, &
    file=__FILE__)) &
    return  ! bail out

  ! set entry point for methods that require specific implementation
  call NUOPC_CompSetEntryPoint(model, ESMF_METHOD_INITIALIZE, &
    phaseLabelList=(/"IPDv00p1"/), userRoutine=InitializeP1, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
    line=__LINE__, &
    file=__FILE__)) &
    return  ! bail out
  call NUOPC_CompSetEntryPoint(model, ESMF_METHOD_INITIALIZE, &
    phaseLabelList=(/"IPDv00p2"/), userRoutine=InitializeP2, rc=rc)
  if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
```

NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
Phases	
Initialize Phase Definition (v00)	
Initialize Phase Definition (v01)	
Initialize Phase Definition (v02)	
Initialize Phase Definition (v03)	
Initialize Phase Definition (v04)	
Specializations	
SetClock [0..1]	
DataInitialize [0..1]	
Create Uniform Grid [0..1]	
Run	
Finalize	
Phases	
Specializations	
FinalizeModel [0..1]	

Fig. 11: Double-clicking on an element in the NUOPC View outline brings the relevant code segment into focus in the editor.

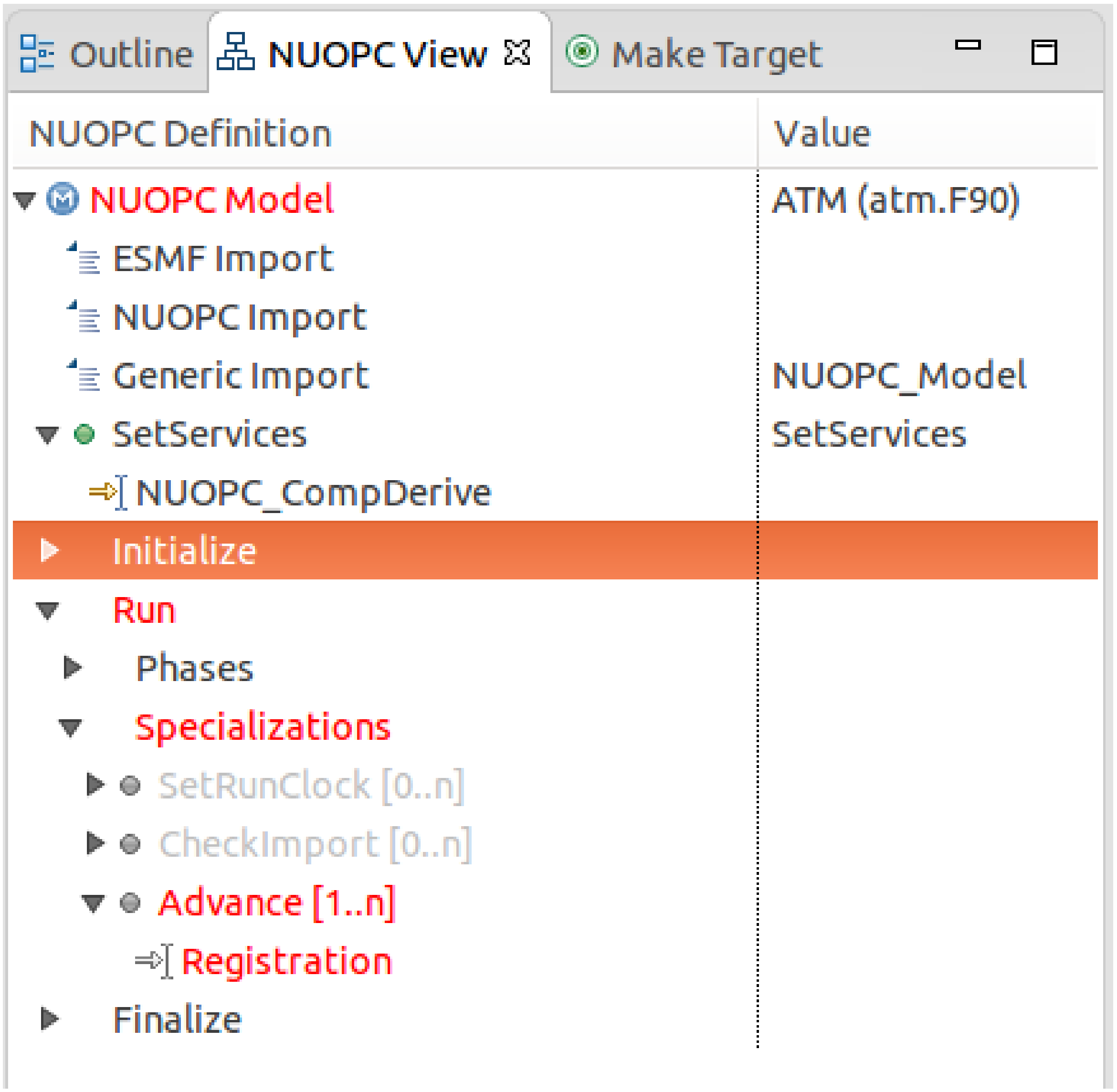
4.2.3 Validation Errors in the NUOPC View

Elements in red in the NUOPC View indicate a validation error. Currently, the validations performed are to check for missing subroutines and API calls required by NUOPC, e.g., a missing initialization phase or a missing specialization point. The NUOPC Reference Manual details, for each type of component, which subroutines are required and which are optional. Red elements do not indicate a Fortran compilation issue, but indicate that NUOPC expects the element to be present and a runtime error will occur without it. The figure below indicates that the *Advance* specialization point could not be found during the reverse engineering procedure. Within NUOPC, *specialization points* are user-provided subroutines that are called by NUOPC. Notice also that parent elements are red all the way to the root of the tree. Therefore, if the root of the tree is red, it indicates a validation issue somewhere below.

To address the issue of the missing Advance element, a new subroutine needs to be added to the code and that subroutine registered in the `SetServices` subroutine. When this is done, the reverse engineering engine will pick up this code and the red elements will disappear. The section *Generate NUOPC-compliant Code* explains how to use Cupid to generate skeleton code for missing elements.

Note: Cupid's reverse engineering and validation engines are based on static source code analysis. The engine depends on an internal program database (Virtual Program Graph or VPG) provided by the Photran plugin for Eclipse.

There are limitations to static analysis giving rise to false negatives–i.e., reporting a validation issue when in fact the NUOPC component will behave correctly. For example, in some cases the reverse engineering engine expects



NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
Run	
Phases	
Specializations	
SetRunClock [0..n]	
CheckImport [0..n]	
Advance [1..n]	
Registration	
Finalize	

Fig. 12: The Advance element is red because it could not be found by the reverse engineering engine.

NUOPC API calls to appear within a given subroutine, say SetServices. In reality, the required API call may appear in a different subroutine called by SetServices or even several levels down in the call tree. Cupid does not currently perform a full control flow analysis to find NUOPC calls because it is an expensive operation. And, even control flow analysis is limited due to conditional logic in the code that depends on the state of the program at runtime.

Cupid, therefore, is fundamentally limited by the realities of static analysis. However, most NUOPC caps have a very similar structure with a fair amount of boilerplate code, so we expect that most codes will be correctly reverse engineered.

4.3 Generate NUOPC-compliant Code

Cupid's code generation facilities make it easier to write the code for a NUOPC cap. A *NUOPC cap* acts as a kind of translation layer between your model code and the coupling infrastructure. A NUOPC cap is implemented as a Fortran module containing a set of subroutines. Cupid is capable of generating NUOPC Model caps, NUOPC Drivers, and NUOPC Mediators. The code generator can create new Fortran modules for each of these components in new files, or the code generator can insert snippits of code into an existing file after it has been reverse engineered.

There are several options for generating code:

- If there is an existing NUOPC component cap, it should be reverse engineered first as described in *Reverse Engineer a NUOPC Cap*. Then, using context menus in the NUOPC View, new code can be generated and inserted in-place. This is the right procedure to use, for example, if you need to add an additional specialization point subroutine to an existing cap.
- If there is no existing NUOPC code, a template can be generated for NUOPC Model caps, NUOPC Drivers, and NUOPC Mediators. This is the best option if you have an existing model and need to create a cap so that it can be used in NUOPC-based coupled systems.
- An entire skeleton NUOPC coupled application can be generated, including a main program and Makefile. This is covered in the *Generate Skeleton Code for a Complete NUOPC Coupled Application* section.

The sections below describe the first two generation options above.

See also:

This user guide is not a comprehensive guide to what comprises a NUOPC cap. For a gentle introduction to NUOPC and what is required in a NUOPC cap, please see the Building a NUOPC Model document.

4.3.1 Generate Code In-Place in an Existing NUOPC component

If you need to modify code in an existing NUOPC component (Model cap, Driver, or Mediator), you should first open up the file so that the reverse engineered outline is shown in the NUOPC View. In the following scenario, let's assume you have an existing NUOPC Model cap for a atmospheric model, but it is missing the required Advance specialization point. This is the subroutine that should call into your model's run phase to take a time step. In the NUOPC View, right-click (CTRL-click on Mac) on the *parent* element of the element you would like to generate. The context menu will show you all code generation options currently available.

In the context menu, select the element to generate, in this case **Generate Advance**. The requested element will be added to the outline and the corresponding code generated in the editor. Often, the addition of one element results in inserting several code fragments. In the case of the Advance element, a new subroutine is added, a new import is added to the `NUOPC_Model` use statement, and a call to `NUOPC_CompSpecialize` is added in the `SetServices` subroutine. After the code generator runs, yellow markers are added to the vertical bar to the right of the code editor to indicate where new code was added. Clicking on one of the markers highlights the generated code.

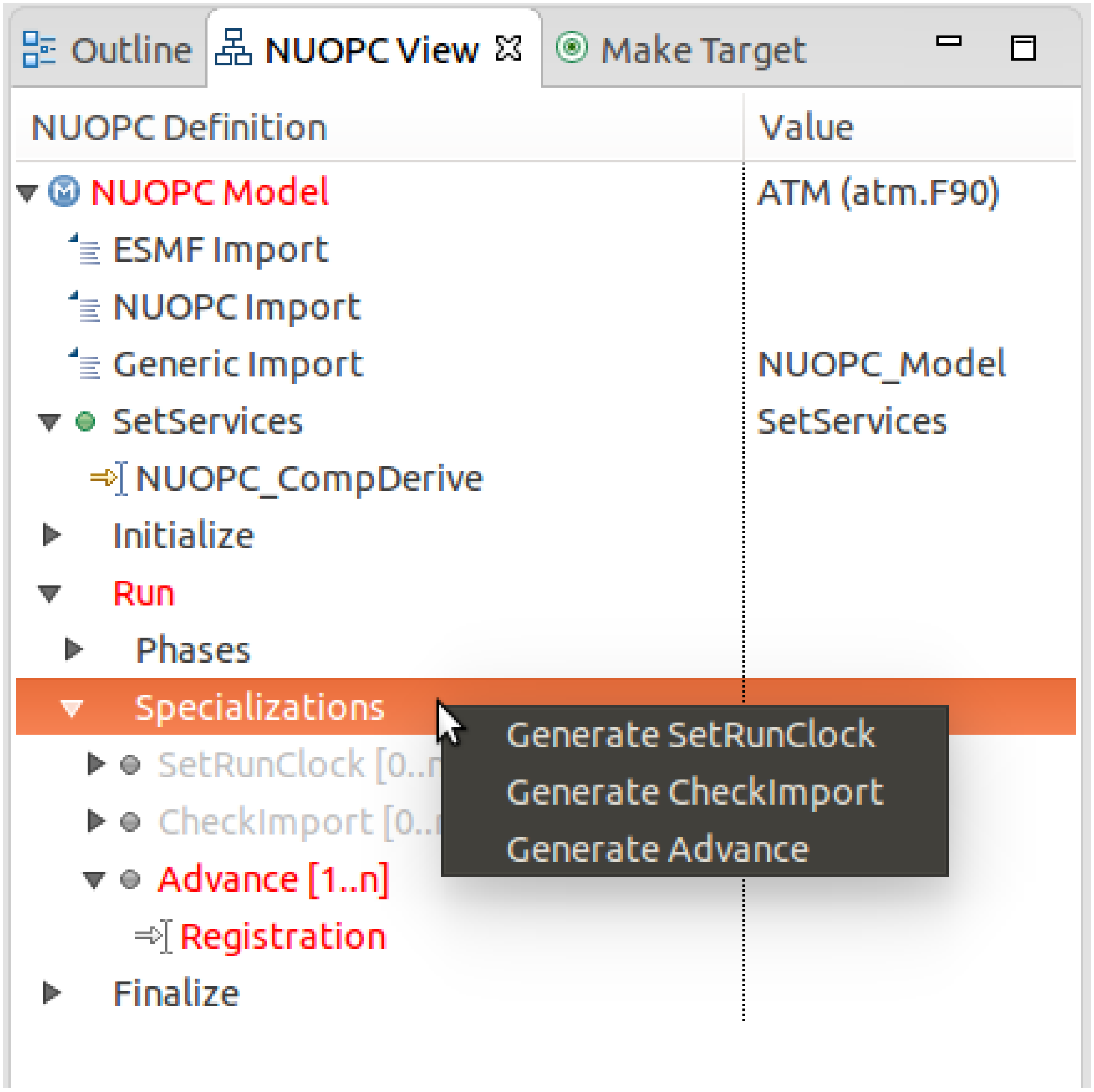
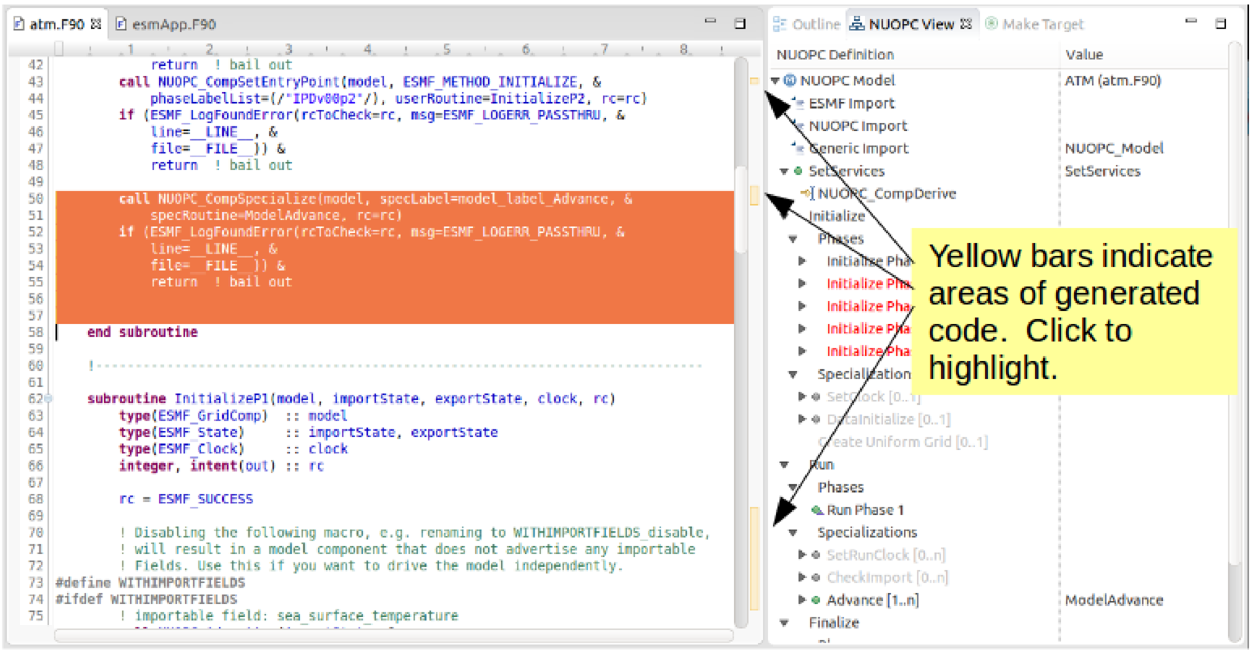


Fig. 13: Right-clicking on an element shows a context menu with the available options for code generation.



atm.F90 | esmApp.F90

```fortran
            return  ! bail out
        call NUOPC_CompSetEntryPoint(model, ESMF_METHOD_INITIALIZE, &
            phaseLabelList=(/"IPDv00p2"/), userRoutine=InitializeP2, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
            line=__LINE__, &
            file=__FILE__)) &
            return  ! bail out

        call NUOPC_CompSpecialize(model, specLabel=model_label_Advance, &
            specRoutine=ModelAdvance, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
            line=__LINE__, &
            file=__FILE__)) &
            return  ! bail out


    end subroutine

    !-----------------------------------------------------------------------------

    subroutine InitializeP1(model, importState, exportState, clock, rc)
        type(ESMF_GridComp)  :: model
        type(ESMF_State)     :: importState, exportState
        type(ESMF_Clock)     :: clock
        integer, intent(out) :: rc

        rc = ESMF_SUCCESS

        ! Disabling the following macro, e.g. renaming to WITHIMPORTFIELDS_disable,
        ! will result in a model component that does not advertise any importable
        ! Fields. Use this if you want to drive the model independently.
#define WITHIMPORTFIELDS
#ifdef WITHIMPORTFIELDS
        ! importable field: sea_surface_temperature
```

Outline | NUOPC View | Make Target

NUOPC Definition	Value
NUOPC Model	ATM (atm.F90)
ESMF Import	
NUOPC Import	
Generic Import	NUOPC_Model
SetServices	SetServices
NUOPC_CompDerive	
Initialize	
Phases	
Initialize Pha	
Initialize Pha	
Initialize Pha	
Initialize Pha	
Initialize Pha	
Specialization	
SetClock [0..1]	
DataInitialize [0..1]	
Create Uniform Grid [0..1]	
Run	
Phases	
Run Phase 1	
Specializations	
SetRunClock [0..n]	
CheckImport [0..n]	
Advance [1..n]	ModelAdvance
Finalize	

Yellow bars indicate areas of generated code. Click to highlight.

Fig. 14: Yellow markers in the vertical bar next to the code editor indicate which code was generated during the *last* code generation action.

The generated code will compile as is, although it almost always requires additional customization to complete the implementation. In the case of the Advance subroutine just generated, additional code is needed to call into the underlying model's time step routine. This clearly cannot be generated automatically because it is model-dependent. Therefore a typical workflow will start with a code generation action as just described, followed by filling in any model-specific implementation. This will continue until all required initialization phases are complete and all specialization points have been implemented.

4.3.2 Generate a NUOPC Model cap, NUOPC Driver, or NUOPC Mediator from Scratch

Templates for NUOPC Model caps, NUOPC Drivers, NUOPC Mediators can be generated from scratch. This option is available from the context menu in the Project Explorer. Right-click (CTRL-click on Mac) on a folder in a Fortran project and select **New** from the context menu and you will see the three options as shown below.

You will be prompted to enter the name of the component. Click OK and a new Fortran file named <COMPONENT>.F90 will appear in the folder (where <COMPONENT> is the name you provided). The file will also automatically open in the editor and you will see the outline in the NUOPC View. At this point the template can be customized by manually adding code and/or generating code fragments from the NUOPC View outline as described above.

To compile the code, you will need to modify your model's existing build system to include the new .F90 file.

4.4 Generate Skeleton Code for a Complete NUOPC Coupled Application

A good way to learn about how NUOPC coupling infrastructure works is to build a skeleton application containing all of the "plumbing" but with no real science code to keep it small.

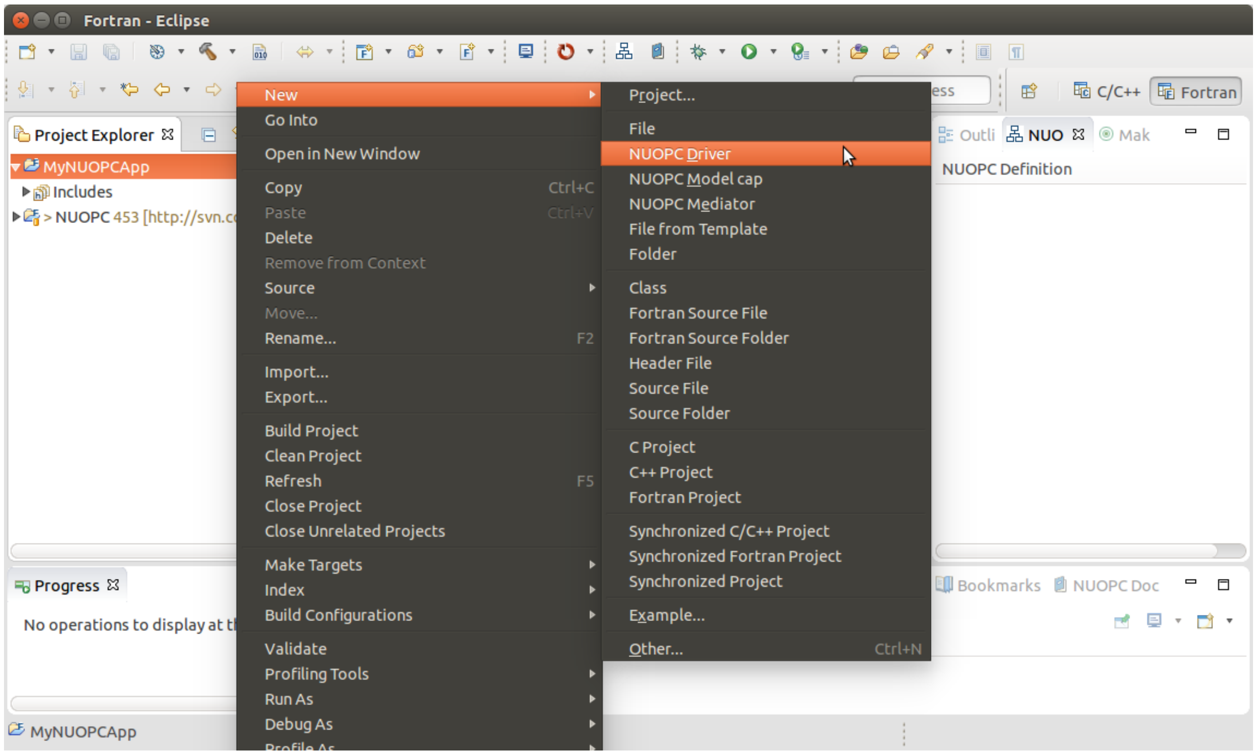
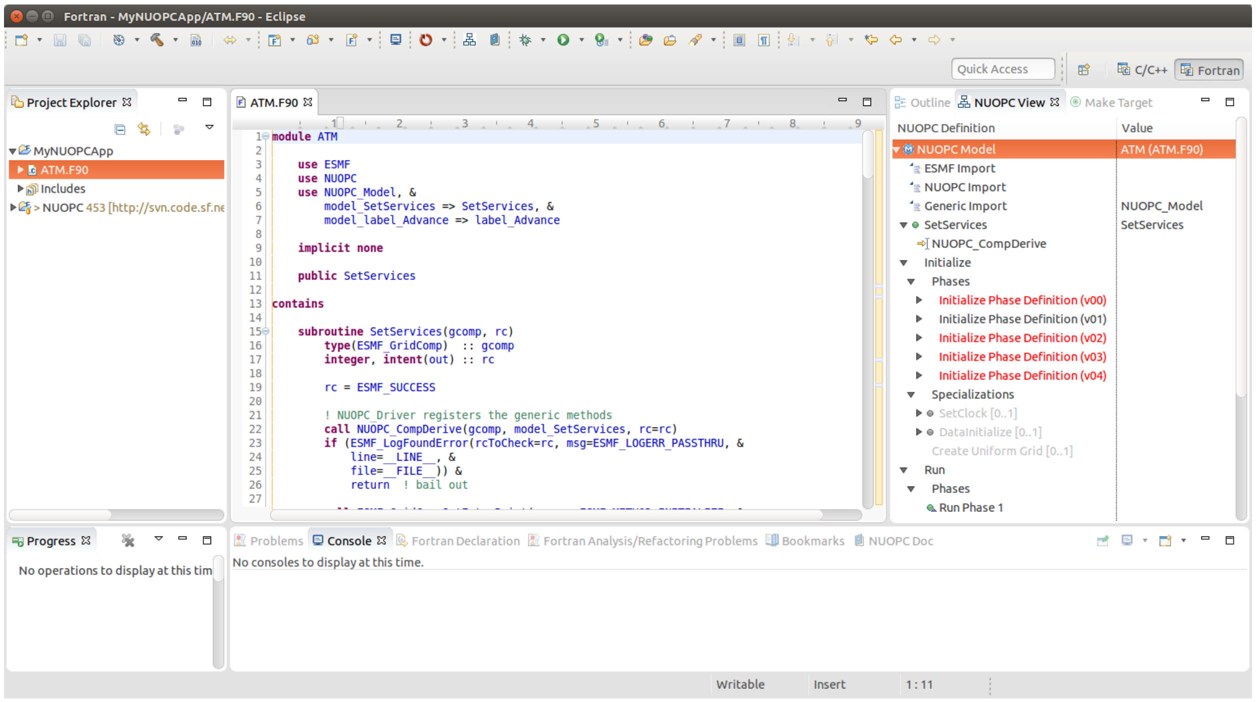


Fig. 15: The Project Explorer context menu with options for generating a NUOPC Model cap, a NUOPC Driver, or a NUOPC Mediator.

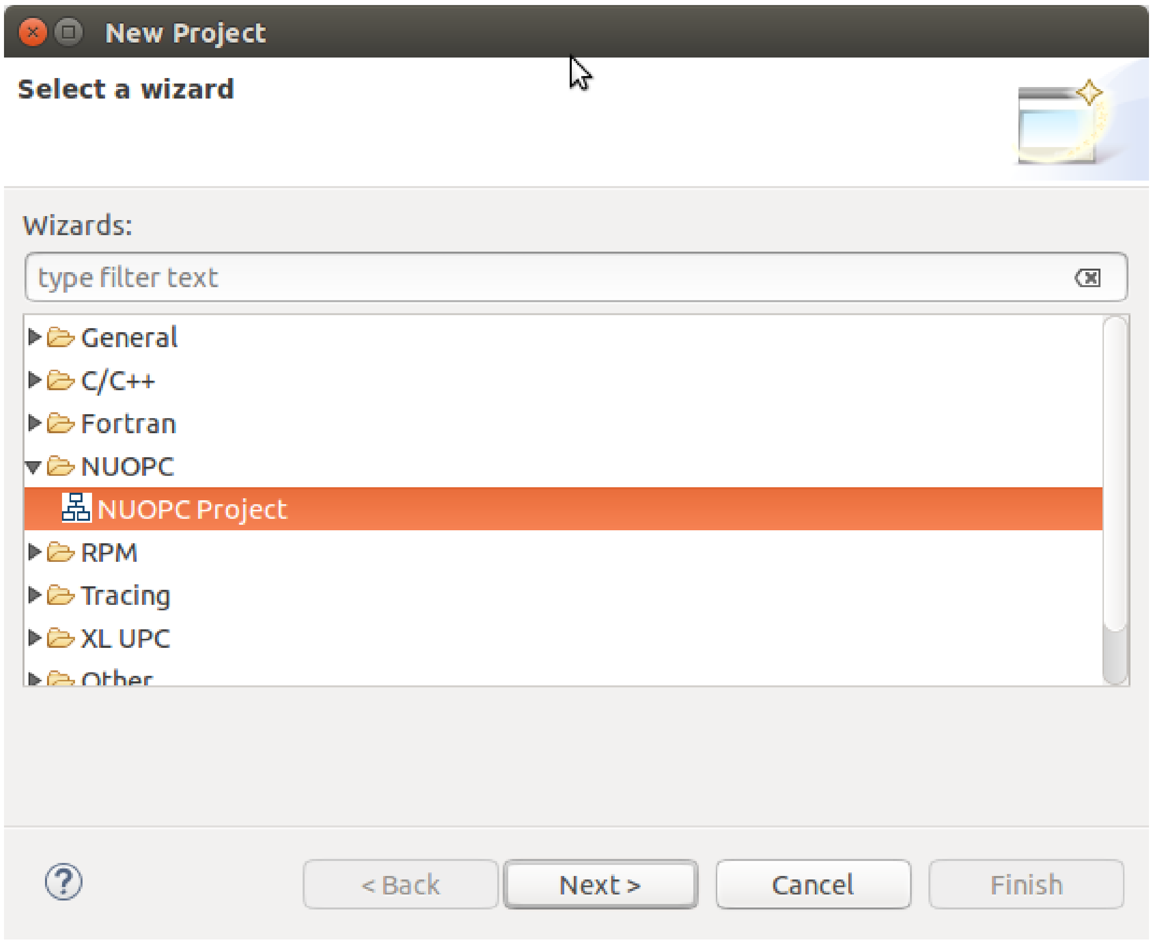


```
module ATM

   use ESMF
   use NUOPC
   use NUOPC_Model, &
      model_SetServices => SetServices, &
      model_label_Advance => label_Advance

   implicit none

   public SetServices

contains

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

      rc = ESMF_SUCCESS

      ! NUOPC_Driver registers the generic methods
      call NUOPC_CompDerive(gcomp, model_SetServices, rc=rc)
      if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHRU, &
         line=__LINE__, &
         file=__FILE__)) &
         return  ! bail out
```

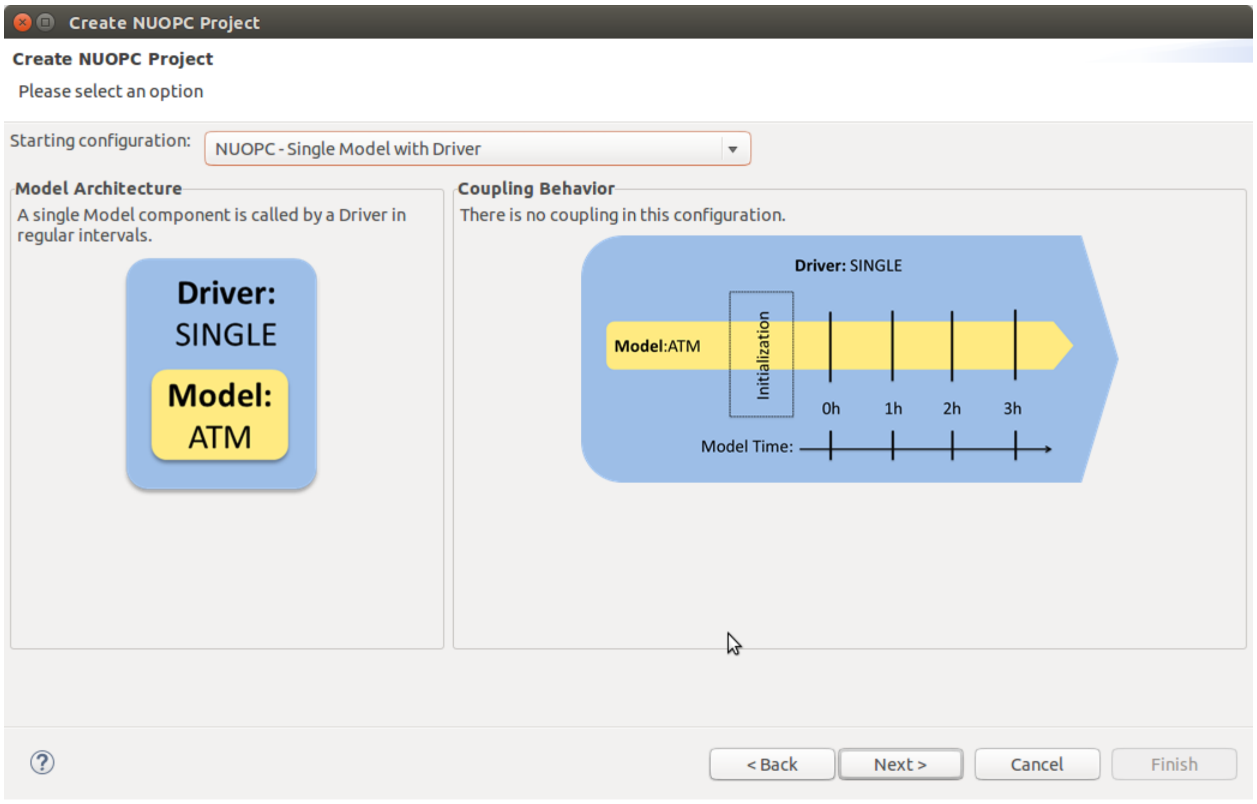
No consoles to display at this time.

Fig. 16: A NUOPC Model cap template.

Create a new NUOPC project using the NUOPC Project wizard. Select **File -> New -> Project...** from the menu. Select the NUOPC Project option under the NUOPC folder and click Next.



On the next screen, select a starting configuration for the skeleton NUOPC application. Ideally, you should find a configuration that looks something like the actual coupled application you are building.


Create NUOPC Project

Create NUOPC Project

Please select an option

Starting configuration: NUOPC - Single Model with Driver

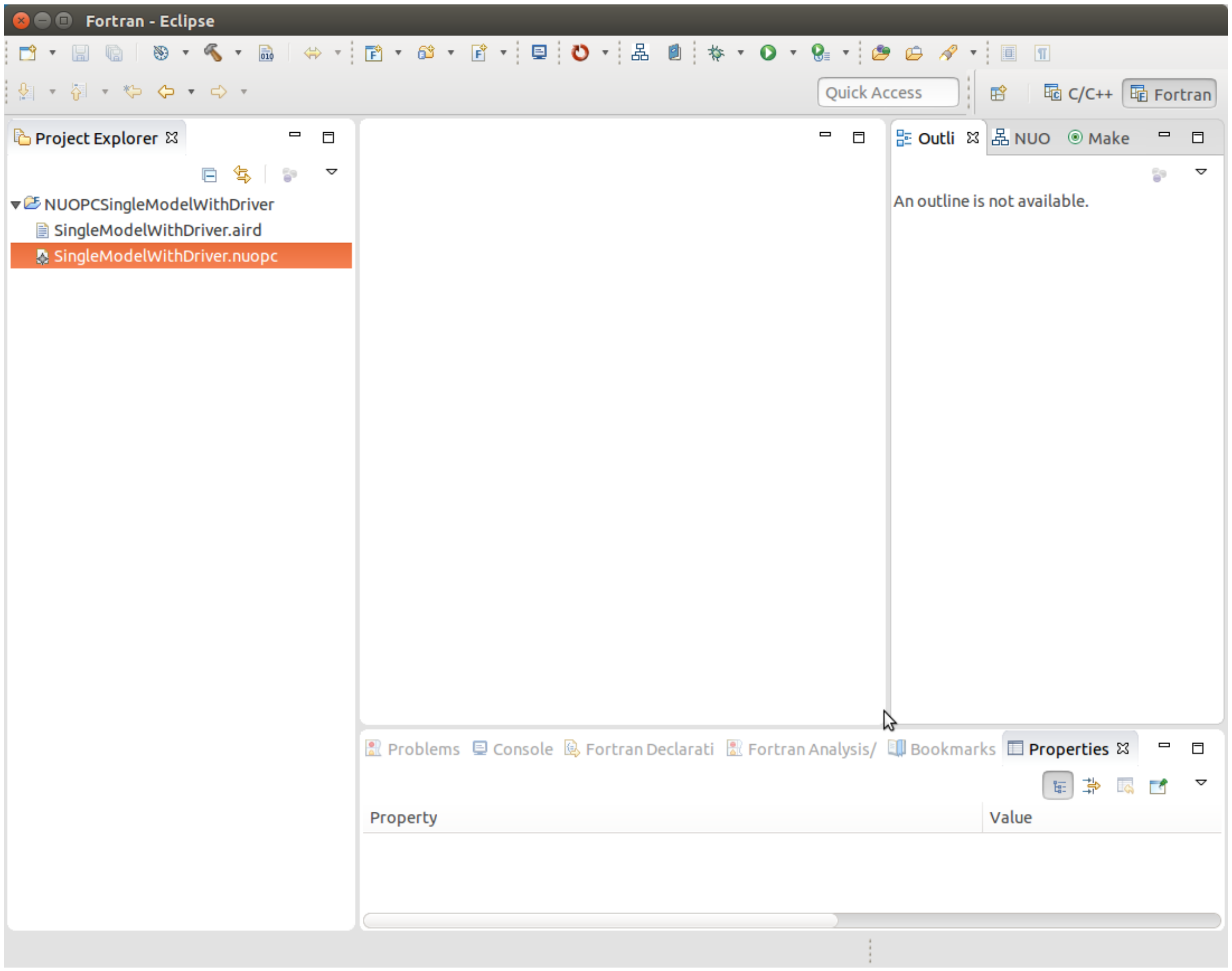
Model Architecture

A single Model component is called by a Driver in regular intervals.

Coupling Behavior

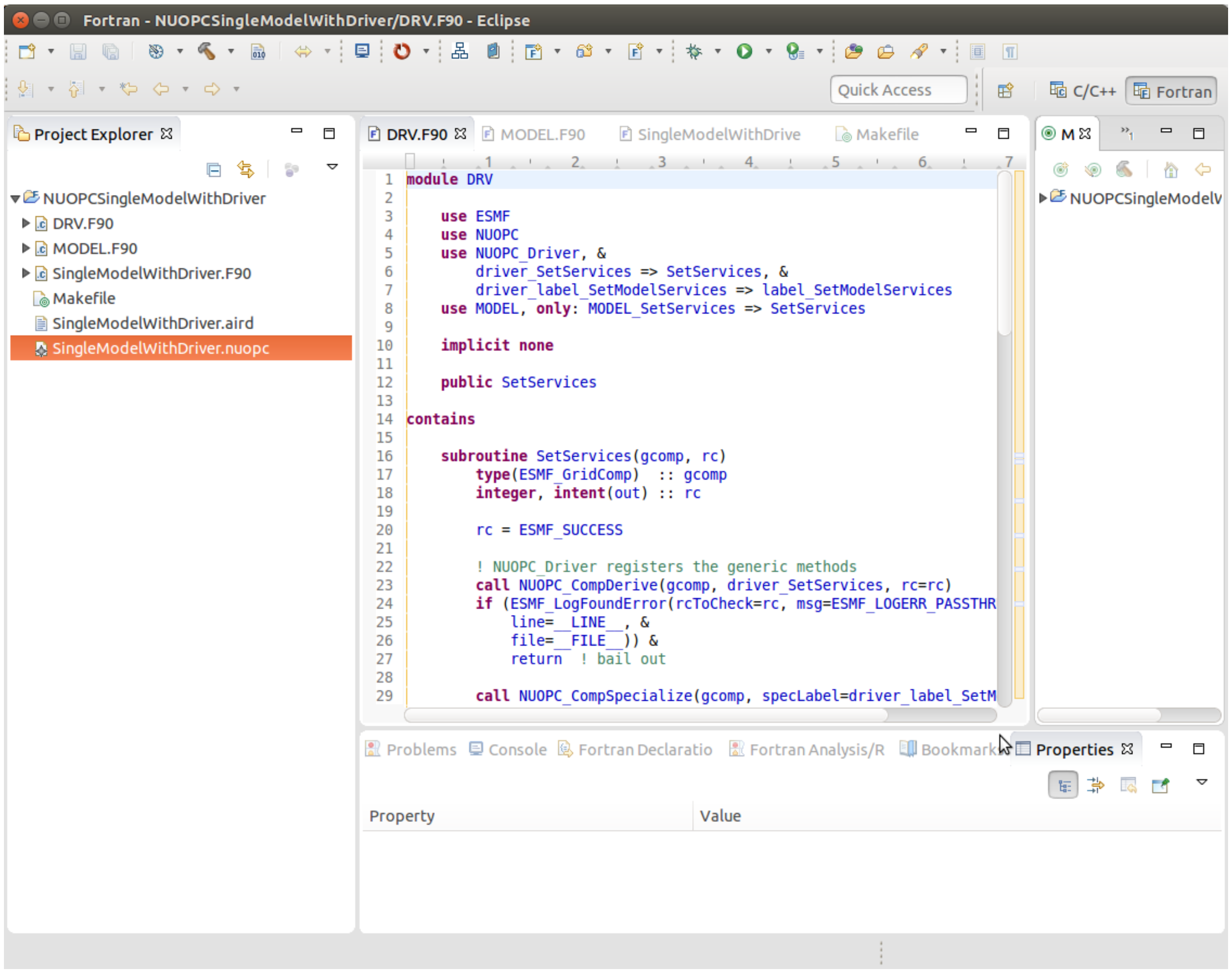
There is no coupling in this configuration.

On the final screen of the wizard, type in a project name and click Finish. The new project will be created. Initially, the project will contain a .nuopc file which is a configuration file describing the coupled system.



To generate all the NUOPC code for the system, right-click (CTRL-click on Mac) on the .nuopc file and select **NUOPC -> Generate NUOPC code** from the context menu. The code for the NUOPC skeleton application will be generated. This includes:

- A NUOPC cap for each Model component
- A NUOPC Mediator, if present in the configuration
- A NUOPC Driver
- A top-level main program
- A makefile



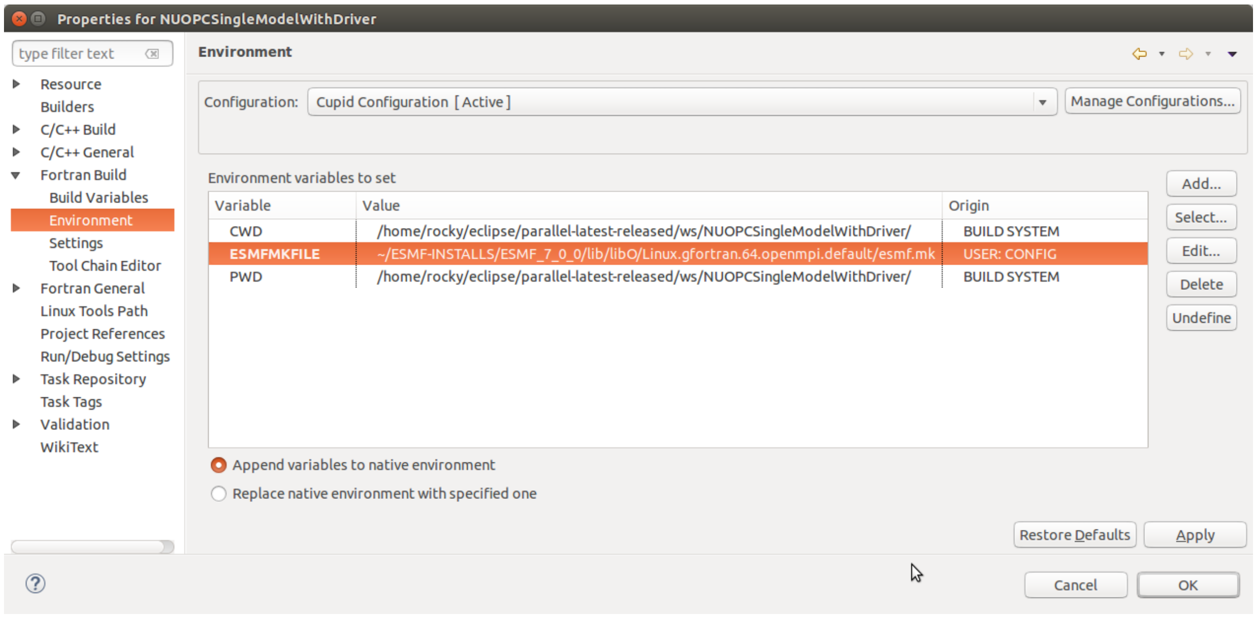
```
module DRV

    use ESMF
    use NUOPC
    use NUOPC_Driver, &
        driver_SetServices => SetServices, &
        driver_label_SetModelServices => label_SetModelServices
    use MODEL, only: MODEL_SetServices => SetServices

    implicit none

    public SetServices

contains

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

        rc = ESMF_SUCCESS

        ! NUOPC_Driver registers the generic methods
        call NUOPC_CompDerive(gcomp, driver_SetServices, rc=rc)
        if (ESMF_LogFoundError(rcToCheck=rc, msg=ESMF_LOGERR_PASSTHR
            line=__LINE__, &
            file=__FILE__)) &
            return  ! bail out

        call NUOPC_CompSpecialize(gcomp, specLabel=driver_label_SetM
```

4.4.1 Build the Skeleton Application Locally

The generated code can now be built using make and the generated Makefile. To build on the same system that Eclipse is running (this is the easiest way), first ensure that ESMF v7 is installed.

The environment variable ESMFMKFILE needs to be set to the location of the esmf.mk file in the ESMF installation directory. It is in the same directory with the ESMF library file(s). (More info on the esmf.mk file is available in the ESMF User Guide.)

To set the ESMFMKFILE environment variable in Eclipse, right click on the project folder in the Project Explorer and select **Properties** from the context menu. Select **Fortran Build -> Environment** in the list on the left and add a new environment variable. Set the name to ESMFMKFILE and the value to the location of the esmf.mk file on your system. Click OK when done.



Environment

Configuration: Cupid Configuration [Active]   Manage Configurations...

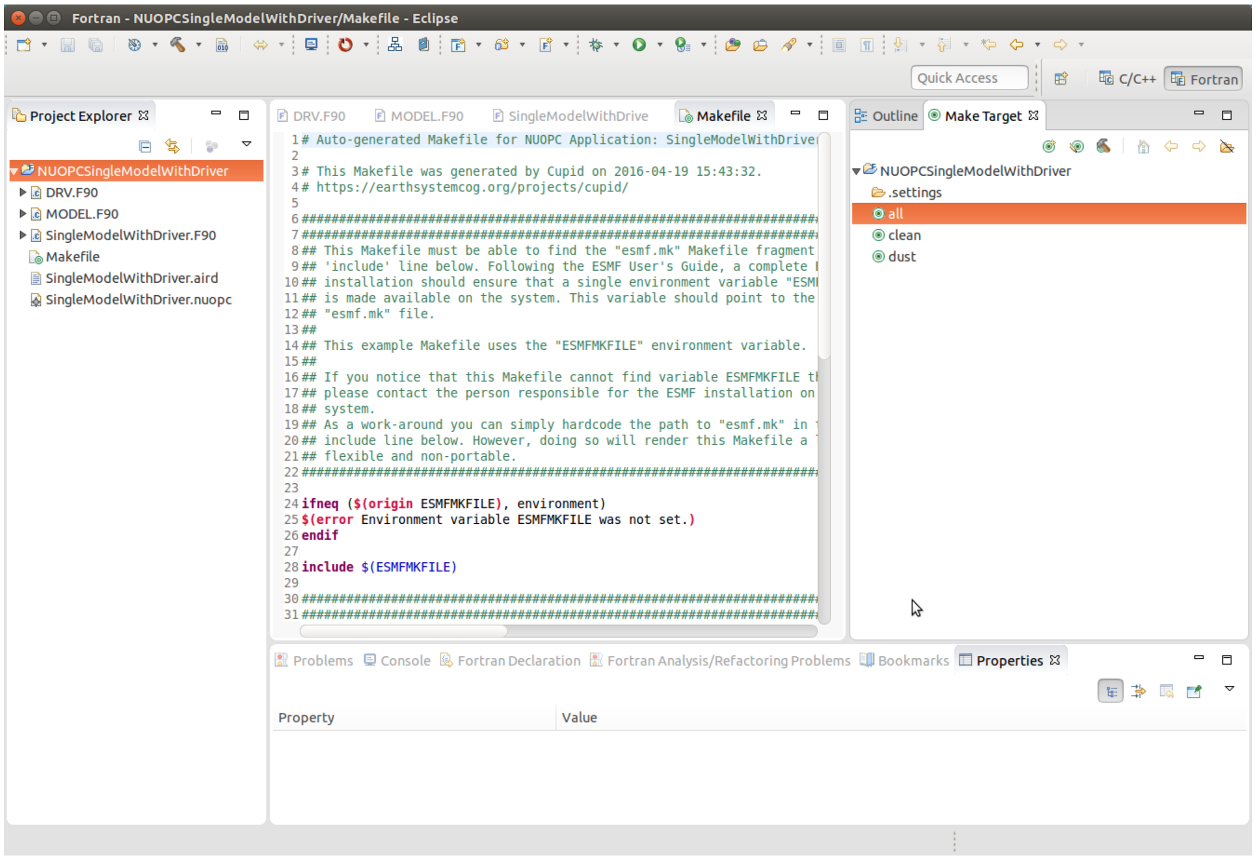
Environment variables to set

Variable	Value	Origin
CWD	/home/rocky/eclipse/parallel-latest-released/ws/NUOPCSingleModelWithDriver/	BUILD SYSTEM
ESMFMKFILE	~/ESMF-INSTALLS/ESMF_7_0_0/lib/libO/Linux.gfortran.64.openmpi.default/esmf.mk	USER: CONFIG
PWD	/home/rocky/eclipse/parallel-latest-released/ws/NUOPCSingleModelWithDriver/	BUILD SYSTEM

[✓ Append variables to native environment]

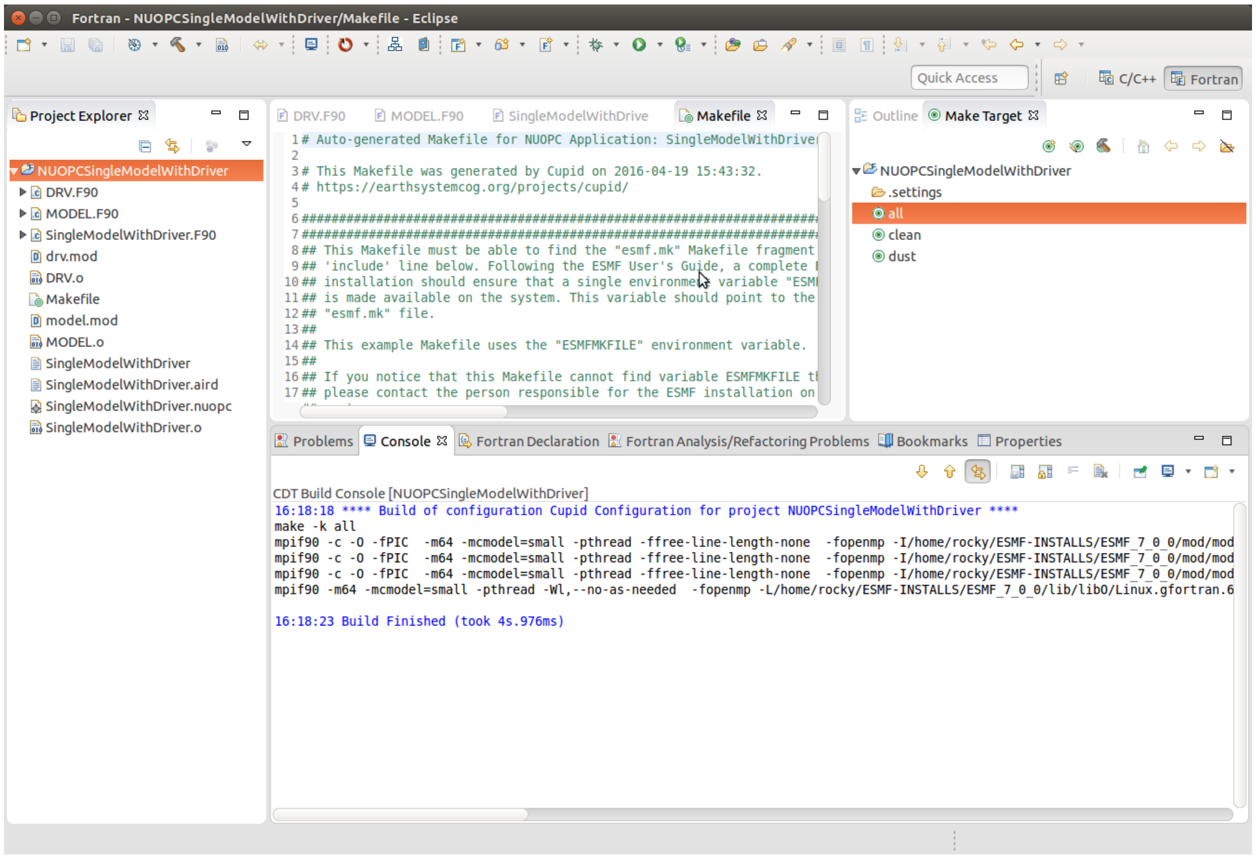
Replace native environment with specified one

To build from within Eclipse, find the Make Target view on the right side and double click the "all" target. If the Make Target view is not shown, you can bring it up by selecting **Window -> Show View -> Make Target** from the menu.


Fortran - NUOPCSingleModelWithDriver/Makefile - Eclipse
Quick Access
C/C++
Fortran
Project Explorer
NUOPCSingleModelWithDriver
DRV.F90
MODEL.F90
SingleModelWithDriver.F90
Makefile
SingleModelWithDriver.aird
SingleModelWithDriver.nuopc
DRV.F90
MODEL.F90
SingleModelWithDrive
Makefile
Outline
Make Target
NUOPCSingleModelWithDriver
.settings
all
clean
dust
Problems
Console
Fortran Declaration
Fortran Analysis/Refactoring Problems
Bookmarks
Properties
Property
Value

```
# Auto-generated Makefile for NUOPC Application: SingleModelWithDrive

# This Makefile was generated by Cupid on 2016-04-19 15:43:32.
# https://earthsystemcog.org/projects/cupid/

######################################################################
######################################################################
## This Makefile must be able to find the "esmf.mk" Makefile fragment
## 'include' line below. Following the ESMF User's Guide, a complete
## installation should ensure that a single environment variable "ESM
## is made available on the system. This variable should point to the
## "esmf.mk" file.
##
## This example Makefile uses the "ESMFMKFILE" environment variable.
##
## If you notice that this Makefile cannot find variable ESMFMKFILE t
## please contact the person responsible for the ESMF installation on
## system.
## As a work-around you can simply hardcode the path to "esmf.mk" in
## include line below. However, doing so will render this Makefile a
## flexible and non-portable.
######################################################################

ifneq ($(origin ESMFMKFILE), environment)
$(error Environment variable ESMFMKFILE was not set.)
endif

include $(ESMFMKFILE)

######################################################################
######################################################################
```

The output from the build will be shown in the Console view at the bottom. The last file built will be the executable and it is typically named the same as the project itself.

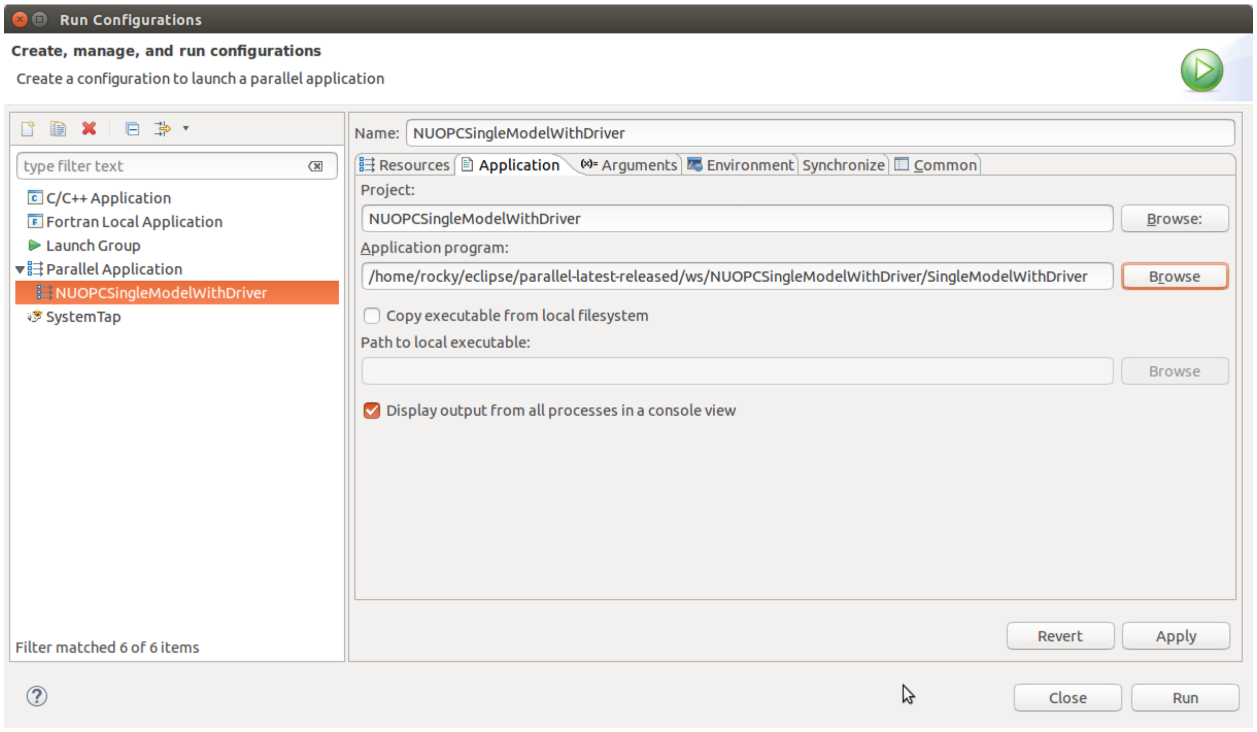


```
# Auto-generated Makefile for NUOPC Application: SingleModelWithDriver

# This Makefile was generated by Cupid on 2016-04-19 15:43:32.
# https://earthsystemcog.org/projects/cupid/

###########################################################################
###########################################################################
## This Makefile must be able to find the "esmf.mk" Makefile fragment
## 'include' line below. Following the ESMF User's Guide, a complete E
## installation should ensure that a single environment variable "ESMI
## is made available on the system. This variable should point to the
## "esmf.mk" file.
##
## This example Makefile uses the "ESMFMKFILE" environment variable.
##
## If you notice that this Makefile cannot find variable ESMFMKFILE tl
## please contact the person responsible for the ESMF installation on
```

```
CDT Build Console [NUOPCSingleModelWithDriver]
16:18:18 **** Build of configuration Cupid Configuration for project NUOPCSingleModelWithDriver ****
make -k all
mpif90 -c -O -fPIC  -m64 -mcmodel=small -pthread -ffree-line-length-none  -fopenmp -I/home/rocky/ESMF-INSTALLS/ESMF_7_0_0/mod/mod
mpif90 -c -O -fPIC  -m64 -mcmodel=small -pthread -ffree-line-length-none  -fopenmp -I/home/rocky/ESMF-INSTALLS/ESMF_7_0_0/mod/mod
mpif90 -c -O -fPIC  -m64 -mcmodel=small -pthread -ffree-line-length-none  -fopenmp -I/home/rocky/ESMF-INSTALLS/ESMF_7_0_0/mod/mod
mpif90 -m64 -mcmodel=small -pthread -Wl,--no-as-needed  -fopenmp -L/home/rocky/ESMF-INSTALLS/ESMF_7_0_0/lib/libO/Linux.gfortran.6

16:18:23 Build Finished (took 4s.976ms)
```

4.4.2 Set up a Parallel Application run and Execute Locally

To execute the application on the same system on which Eclipse is running (again, this is the easiest way), set up a Parallel Application run configuration by selecting **Run -> Run Configurations…** from the menu. The configuration will be dependent on the MPI distribution on your local machine, but you should use the same MPI distribution that was used to compile ESMF. On the Application tab, you need to select the location of the executable that was generated.



Run Configurations

Create, manage, and run configurations

Create a configuration to launch a parallel application

type filter text

- C/C++ Application
- Fortran Local Application
- Launch Group
- Parallel Application
  - NUOPCSingleModelWithDriver
- SystemTap

Filter matched 6 of 6 items

Name: NUOPCSingleModelWithDriver

Resources | Application | Arguments | Environment | Synchronize | Common

Project: NUOPCSingleModelWithDriver

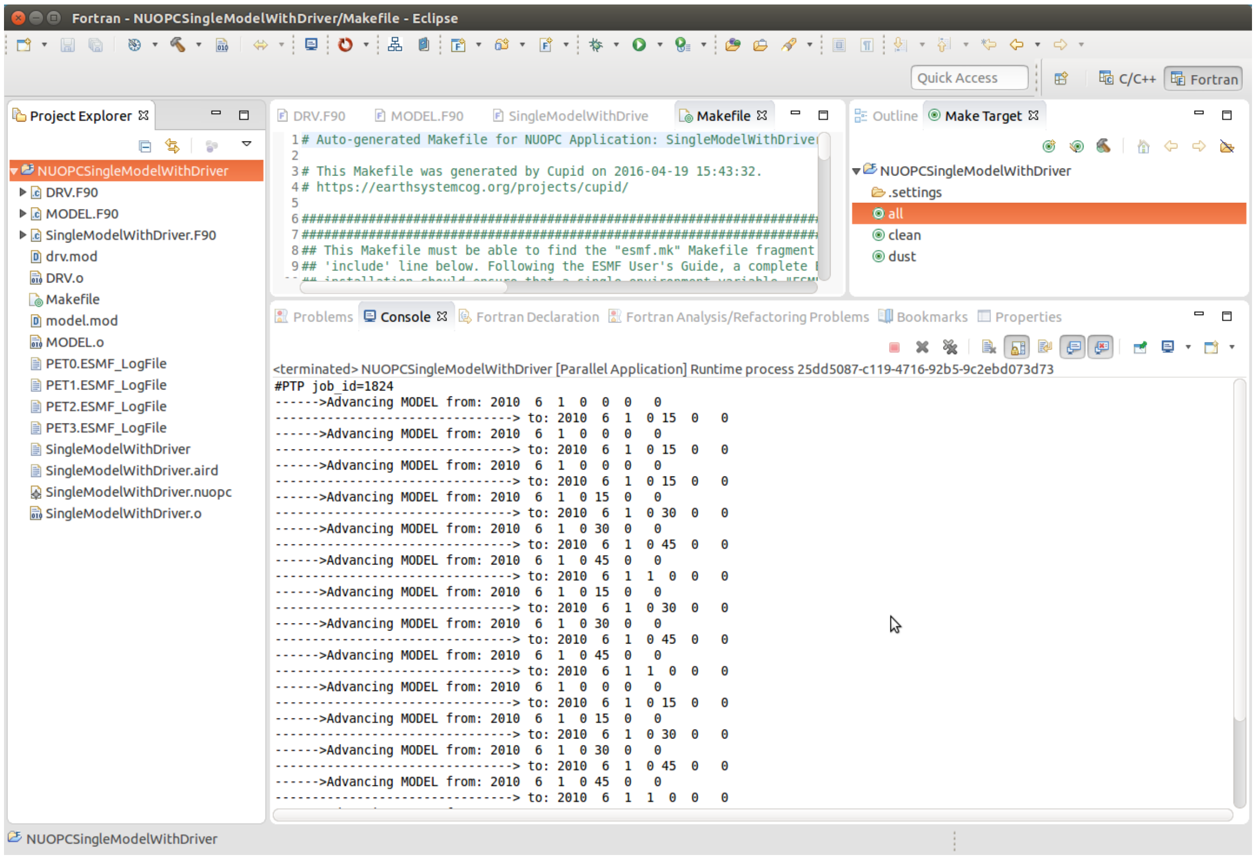
Application program: /home/rocky/eclipse/parallel-latest-released/ws/NUOPCSingleModelWithDriver/SingleModelWithDriver

[ ] Copy executable from local filesystem

Path to local executable:

[✓] Display output from all processes in a console view

After configuring the parallel run, click Run and you will see output from the run in Console. ESMF log files will also be generated, one per process. These are named PETX.ESMF_LogFile. If you do not see the log files immediately after the run, right click on the project folder and select **Refresh** from the context menu.



Fortran - NUOPCSingleModelWithDriver/Makefile - Eclipse

Project Explorer

- NUOPCSingleModelWithDriver
  - DRV.F90
  - MODEL.F90
  - SingleModelWithDriver.F90
  - drv.mod
  - DRV.o
  - Makefile
  - model.mod
  - MODEL.o
  - PET0.ESMF_LogFile
  - PET1.ESMF_LogFile
  - PET2.ESMF_LogFile
  - PET3.ESMF_LogFile
  - SingleModelWithDriver
  - SingleModelWithDriver.aird
  - SingleModelWithDriver.nuopc
  - SingleModelWithDriver.o

DRV.F90 | MODEL.F90 | SingleModelWithDrive | Makefile

```
# Auto-generated Makefile for NUOPC Application: SingleModelWithDrive

# This Makefile was generated by Cupid on 2016-04-19 15:43:32.
# https://earthsystemcog.org/projects/cupid/

#######################################################################
#######################################################################
## This Makefile must be able to find the "esmf.mk" Makefile fragment
## 'include' line below. Following the ESMF User's Guide, a complete 
```

Outline | Make Target

- NUOPCSingleModelWithDriver
  - .settings
  - all
  - clean
  - dust

Problems | Console | Fortran Declaration | Fortran Analysis/Refactoring Problems | Bookmarks | Properties

<terminated> NUOPCSingleModelWithDriver [Parallel Application] Runtime process 25dd5087-c119-4716-92b5-9c2ebd073d73

```
#PTP job_id=1824
------>Advancing MODEL from: 2010  6  1  0  0  0   0
-------------------------------> to: 2010  6  1  0 15  0   0
------>Advancing MODEL from: 2010  6  1  0  0  0   0
-------------------------------> to: 2010  6  1  0 15  0   0
------>Advancing MODEL from: 2010  6  1  0  0  0   0
-------------------------------> to: 2010  6  1  0 15  0   0
------>Advancing MODEL from: 2010  6  1  0 15  0   0
-------------------------------> to: 2010  6  1  0 30  0   0
------>Advancing MODEL from: 2010  6  1  0 30  0   0
-------------------------------> to: 2010  6  1  0 45  0   0
------>Advancing MODEL from: 2010  6  1  0 45  0   0
-------------------------------> to: 2010  6  1  1  0  0   0
------>Advancing MODEL from: 2010  6  1  0 15  0   0
-------------------------------> to: 2010  6  1  0 30  0   0
------>Advancing MODEL from: 2010  6  1  0 30  0   0
-------------------------------> to: 2010  6  1  0 45  0   0
------>Advancing MODEL from: 2010  6  1  0 45  0   0
-------------------------------> to: 2010  6  1  1  0  0   0
------>Advancing MODEL from: 2010  6  1  0  0  0   0
-------------------------------> to: 2010  6  1  0 15  0   0
------>Advancing MODEL from: 2010  6  1  0 15  0   0
-------------------------------> to: 2010  6  1  0 30  0   0
------>Advancing MODEL from: 2010  6  1  0 30  0   0
-------------------------------> to: 2010  6  1  0 45  0   0
------>Advancing MODEL from: 2010  6  1  0 45  0   0
-------------------------------> to: 2010  6  1  1  0  0   0
```

NUOPCSingleModelWithDriver

4.5 Show the NUOPC Reference Manual

The NUOPC Reference Manual can be shown directly within Eclipse so that you do not need to leave the tool to read API documentation. To open the NUOPC documentation viewer, either click on the Show NUOPC Doc View button in the toolbar or from the menu select **Window -> Show View -> Other** and select the *NUOPC Doc* view in the list.

If you select a component in the NUOPC View, the documentation viewer will synchronize with the selected item. For example, if a NUOPC Mediator component is selected in the NUOPC View outline, the documentation viewer will bring that part of the Reference Manual into focus.

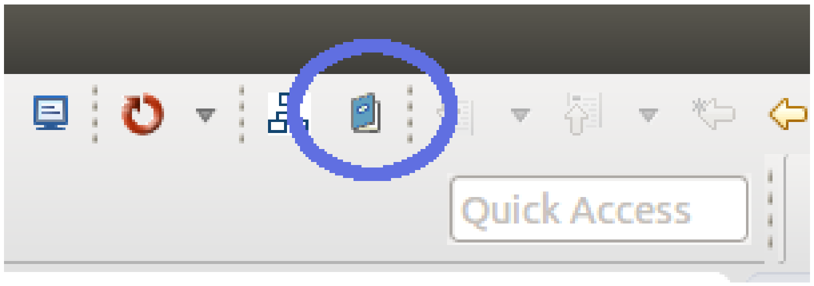
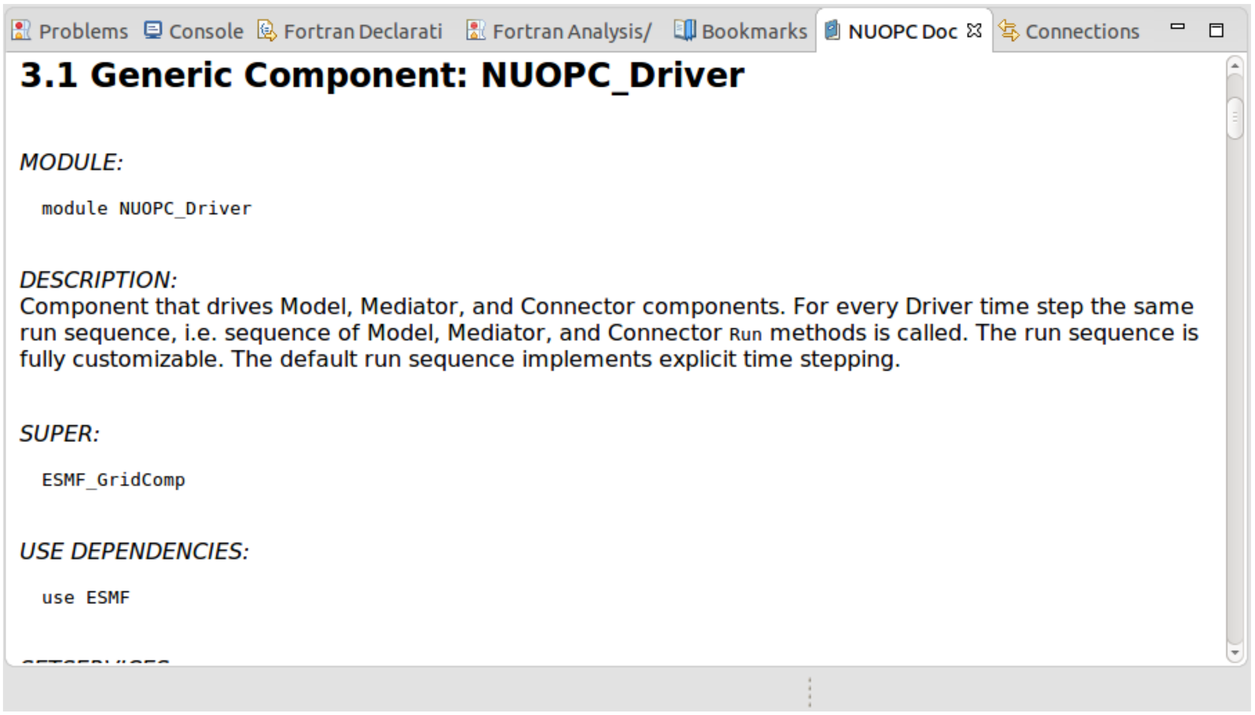


Fig. 17: Click the blue book in the toolbar to show the NUOPC Reference Manual.



3.1 Generic Component: NUOPC_Driver

MODULE:

```
module NUOPC_Driver
```

DESCRIPTION:
Component that drives Model, Mediator, and Connector components. For every Driver time step the same run sequence, i.e. sequence of Model, Mediator, and Connector `Run` methods is called. The run sequence is fully customizable. The default run sequence implements explicit time stepping.

SUPER:

```
ESMF_GridComp
```

USE DEPENDENCIES:

```
use ESMF
```

Fig. 18: The NUOPC Reference Manual is opened in a small browser built into Eclipse.

CHAPTER 5

Search

- search