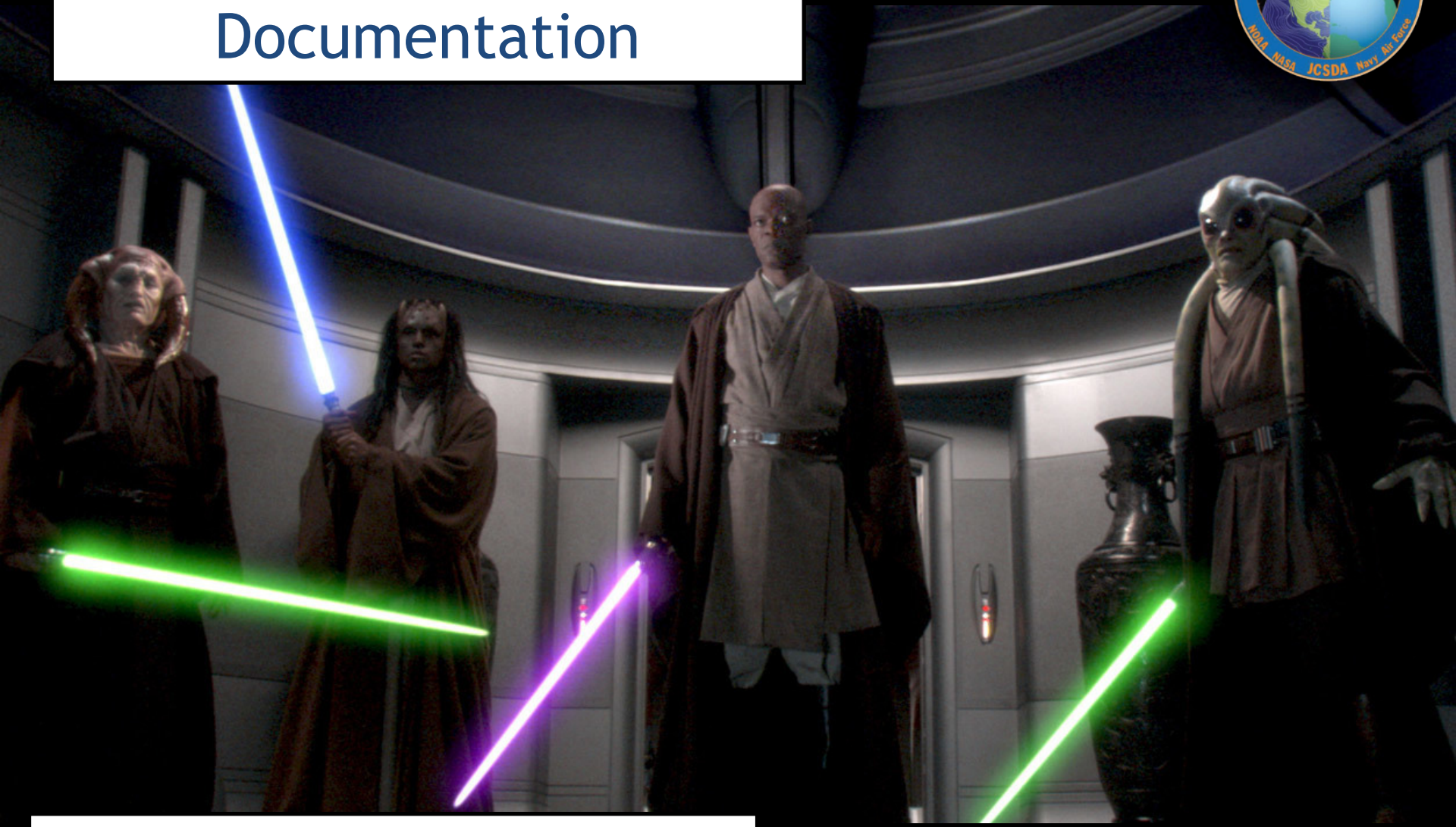
Collaborative Tools 2 Documentation



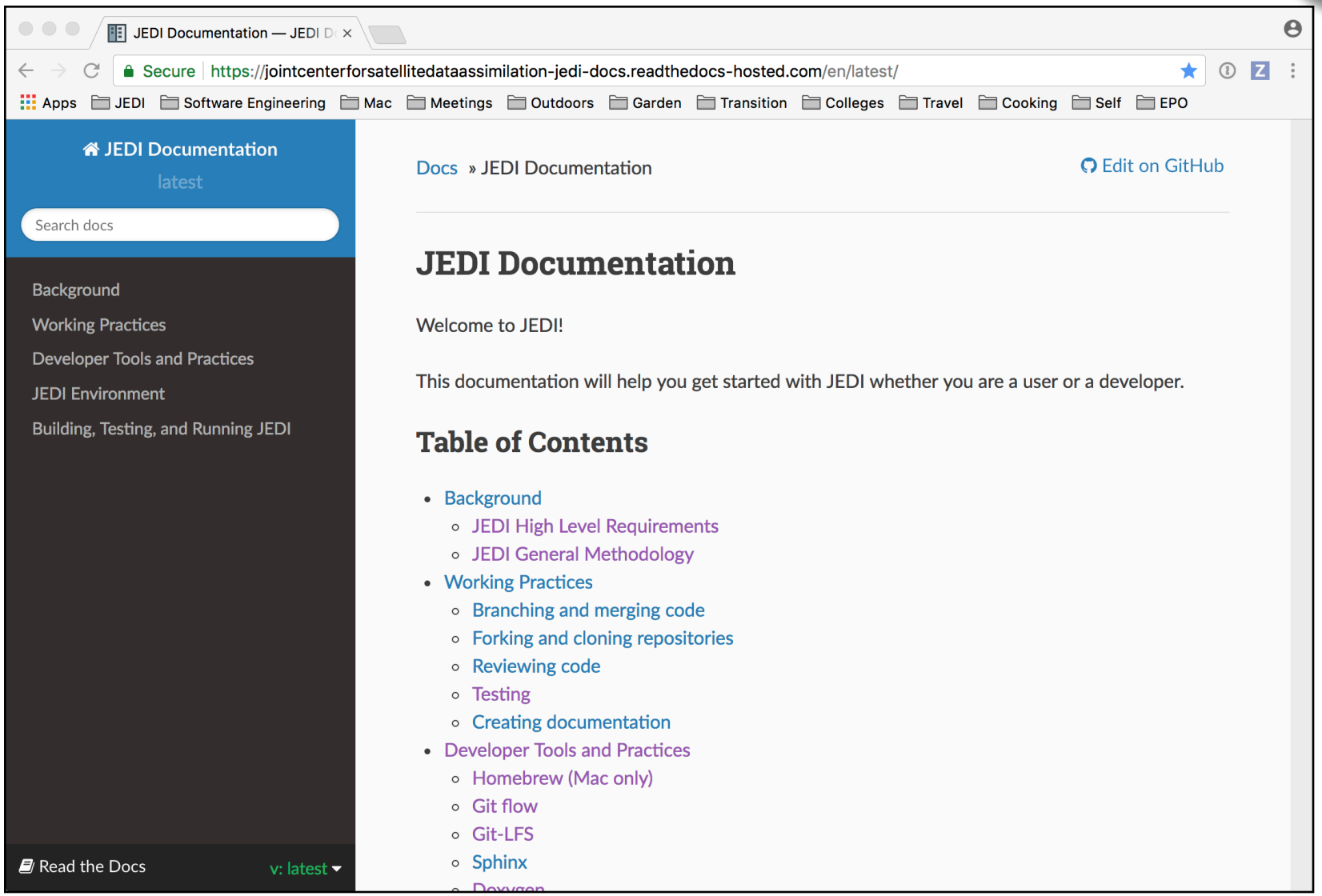
JEDI Academy – 4-7 June 2018

http://starwars.com

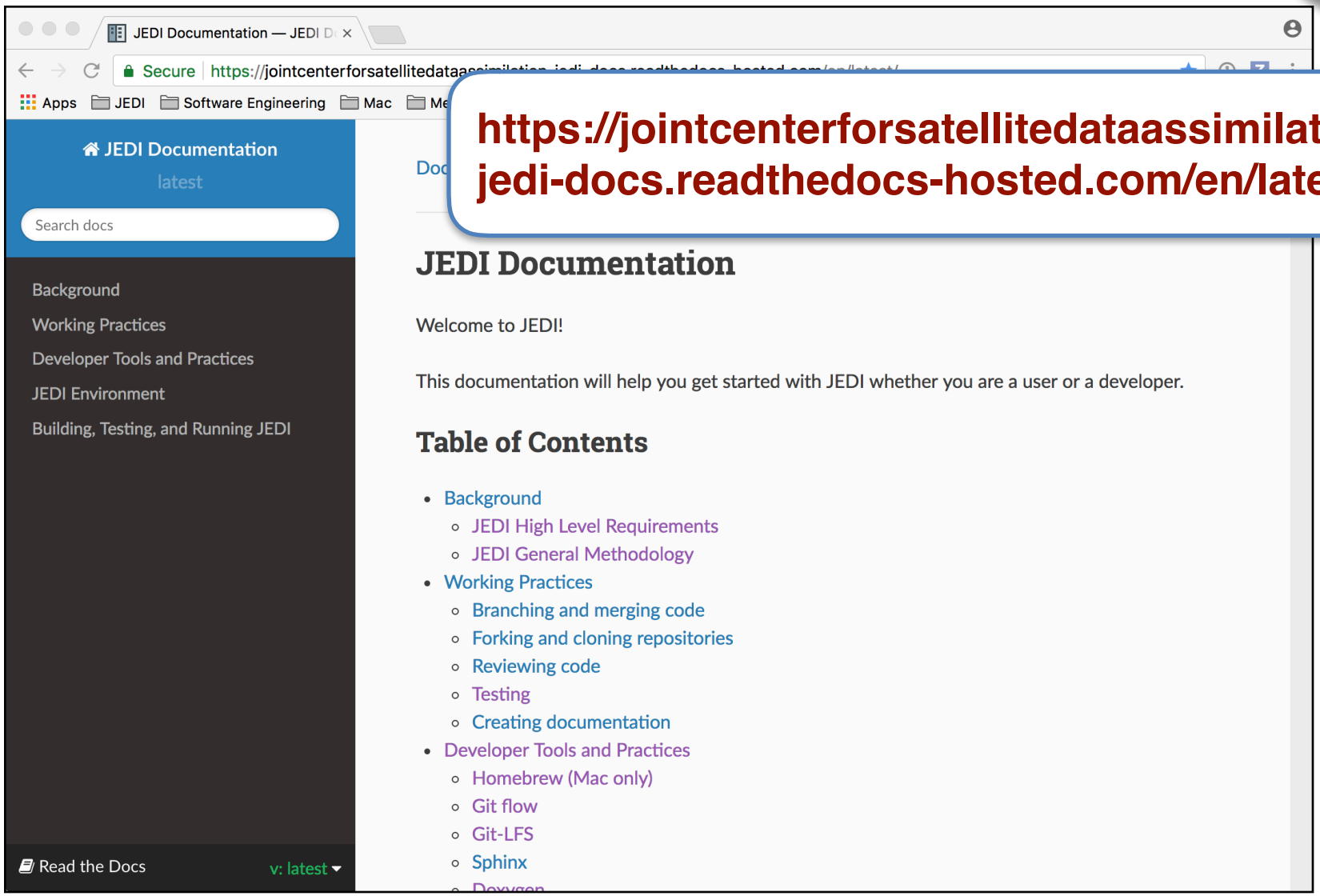
Outline

- ***Sphinx / ReadTheDocs***
  - ***Publicly available***
  - ***Geared toward users as well as developers***
- ***Doxygen***
  - ***Low-level documentation of specific code components (classes, functions, modules, etc)***
  - ***Commands embedded in the code***
  - ***Html output (also optionally pdf, man pages)***
- ***JEDI Wiki***
  - ***Targeted at developers***
  - ***Discussion of current progress, issues***
  - ***Resources for workshops and other events (e.g. code sprints)***

Sphinx/ReadtheDocs

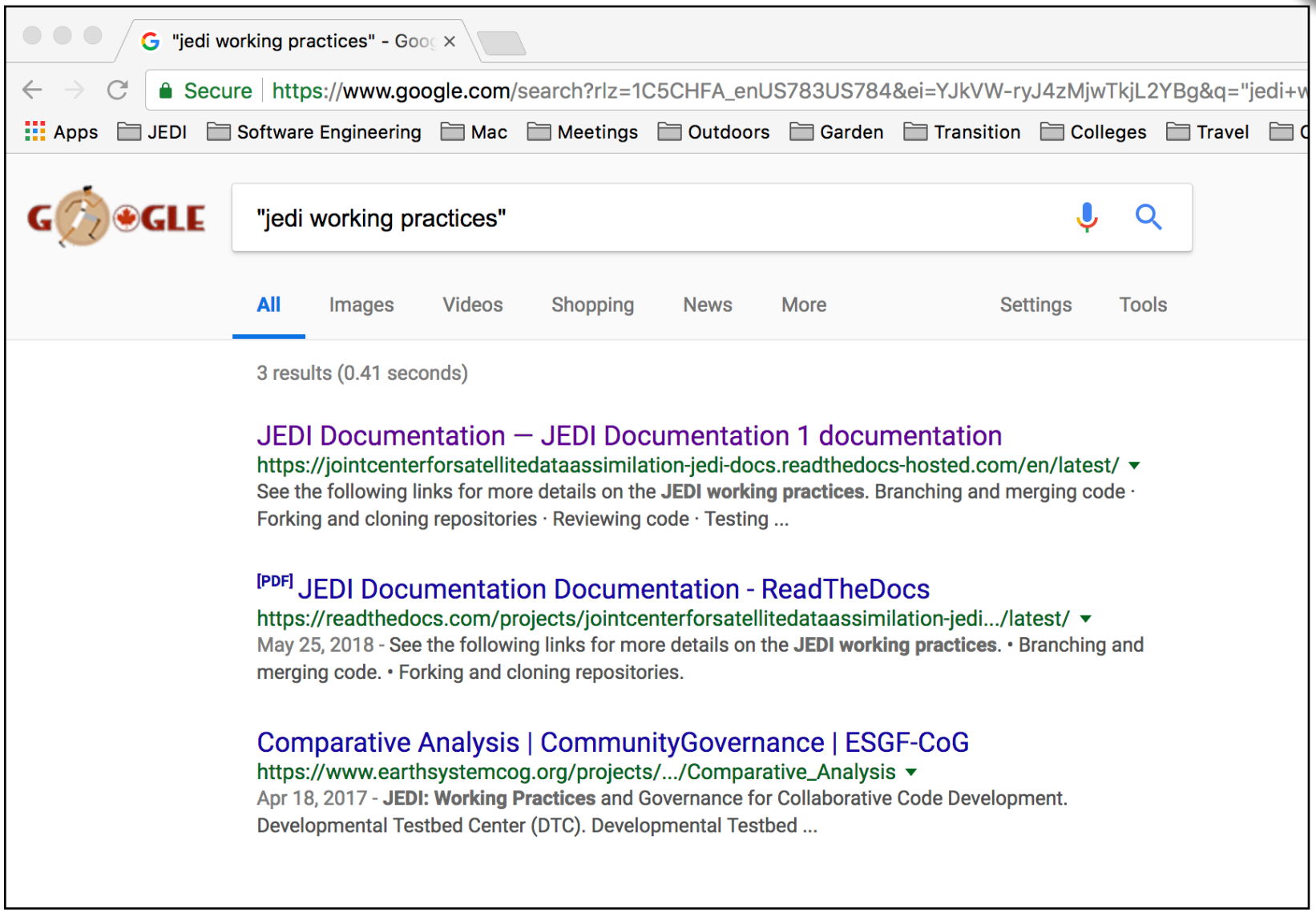


Sphinx/ReadtheDocs

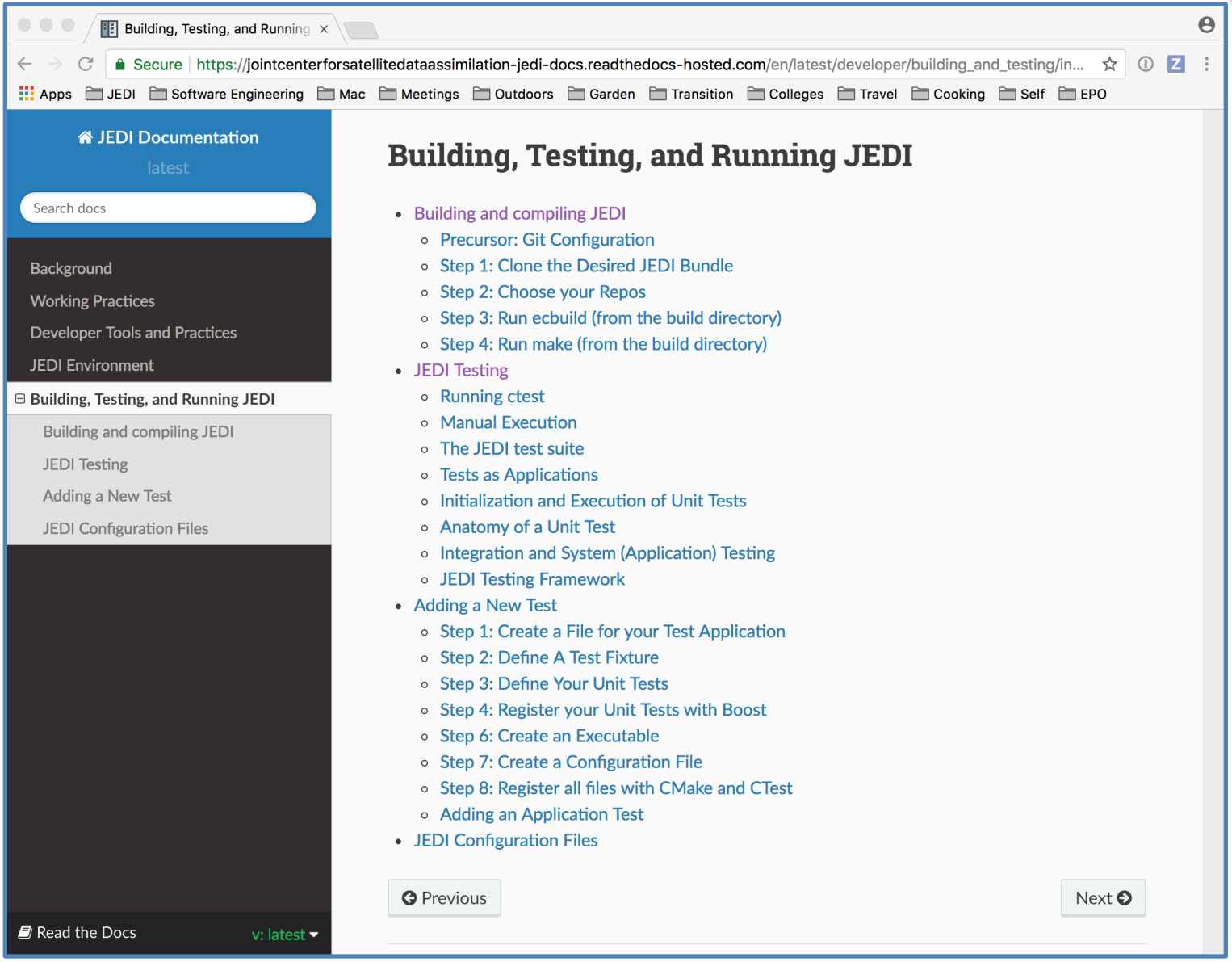


https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/

Sphinx/ReadtheDocs



Sphinx/ReadtheDocs



Sphinx

- ***Sphinx***
  - ***The real workhorse behind the documents***
  - ***Python package***
  - ***Source code written with Restructured text***

- ***Distribution plan***
  - ***ReadtheDocs for now to publish***
  - ***Sphinx Source code on GitHub (jedi-docs)***
  - ***Tagged versions of the doc repos will be linked to JEDI releases***

For more on Sphinx:

[JEDI ReadtheDocs page about Sphinx (in Developer tools and Practices)]

http://www.sphinx-doc.org/en/master/index.html

http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

Doxygen

Used in JEDI for:

- ***Documenting functions and subroutines (C++ and F90)***
- ***Documenting classes and structures (C++ and F90)***
- ***Viewing namespaces and modules***
- ***Generating Class Hierarchies***
- ***Generating Call diagrams***
- ***Any other documentation that involves specific blocks of code***

Whenever you add code to any JEDI Repo, please document it with Doxygen

Documenting C++ Source Code

```
// ----------------------------------------------------------------------------
/*! \brief Example function
*
* \details **myfunction()** takes a and b as arguments and miraculously creates c.
* I could add many more details here if I chose to do so.  I can even make a list:
* * item 1
* * item 2
* * item 3
*
* \param[in] a this is one input parameter
* \param[in] b this is another
* \param[out] c and this is the output
*
* \date A long, long, time ago: Created by L. Skywalker (JCSDA)
*
* \warning This isn't a real function!
*
*/
void myfunction(int& a, int& b, double& c) {
   [...]
```

Documenting Fortran Source Code

```
! ! ------------------------------------------------------------------------------
!> \brief Example function
!!
!! \details **myfunction()** takes a and b as arguments and miraculously creates c.
!! I could add many more details here if I chose to do so.  I can even make a list:
!! * item 1
!! * item 2
!! * item 3
!!
!! \date A long, long, time ago: Created by L. Skywalker (JCSDA)
!!
!! \warning This isn't a real function!
!!
subroutine myfunction(a, b, c)
   integer, intent(in)             :: a !< this is one input parameter
   integer, intent(in)             :: b !< this is another
   real(kind=kind_rea), intent(out) :: c !< and this is the output
   [...]
```

Useful Doxygen Commands

- \brief
- \details
- \param
- \return
- \author
- \date
- \note
- \attention
- \warning
- \bug
- \class <name> [<header-file>]
- \mainpage
- \f$ … \f$ (inline formula)
- \f[… \f] (formula block)
- \em (or * … *)
- \sa (see also)
- \typedef
- \todo
- \version
- \namespace
- […](…) (url)
- \image
- \var
- \throws (exception description)

Many more described here:

https://www.stack.nl/~dimitri/doxygen/manual/commands.html

Doxygen Implementation Plan

- ***User/Developers (this means you!)***
  - ✦Please place appropriate Doxygen comments in source files
  - ✦(optionally) test functionality by compiling with Doxygen config files provided by JEDI team (feel free to customize, but please don't commit your changes)
    - Find Doxyfile (the plan is to have one in the Documents directory of every repo)

      > doxygen
    - View results in html directory

- ***JEDI Core Team***
  - ✦Will supply the Doxyfile config files
  - ✦Will publish html files for develop and master versions of repos (generated automatically, triggered by pull requests)
  - ✦Tagged versions linked to releases
  - ✦Please be patient - We're still working on this

Doxygen Installation (Mac)

> brew install doxygen

You may be prompted to also install Doxywizard and Graphviz - we recommend you say yes to both… If Graphviz does not install for some reason, you can install it manually:

> brew install graphviz

This puts dot in /usr/local/bin
You'll need this for generating graphs

Sample output: “man page”

◆ testStateInterpolation()

template<typename MODEL >

void test::testStateInterpolation ( )

Interpolation test.

testStateInterpolation() tests the interpolation for a given model. The conceptual steps are as follows:

1. Initialize the JEDI **State** object based on idealized analytic formulae
2. Interpolate the **State** variables onto selected "observation" locations using the getValues() method of the **State** object. The result is placed in a JEDI **GeoVaLs** object
3. Compute the correct solution by applying the analytic formulae directly at the observation locations.
4. Assess the accuracy of the interpolation by comparing the interpolated values from Step 2 with the exact values from Step 3

The interpolated state values are compared to the analytic solution for a series of **locations** which includes values optionally specified by the user in the "StateTest" section of the config f
a randomly-generated list of **Nrandom** random locations. Nrandom is also specified by the user in the "StateTest" section of the config file, as is the (nondimensional) tolerence level (**inte
to be used for the tests.

This is an equation:

$$\zeta = \left(\frac{x - x_0}{\lambda} \right)^{2/3}$$

Relevant parameters in the **State* section of the config file include

- **norm-gen** Normalization test for the generated **State**
- **interp_tolerance** tolerance for the interpolation test

Date

April, 2018: M. Miesch (JCSDA) adapted a preliminary version in the feature/interp branch

Warning

Since this model compares the interpolated state values to an exact analytic solution, it requires that the "analytic_init" option be implemented in the model and selected in the "State.StateGenerate" section of the config file.

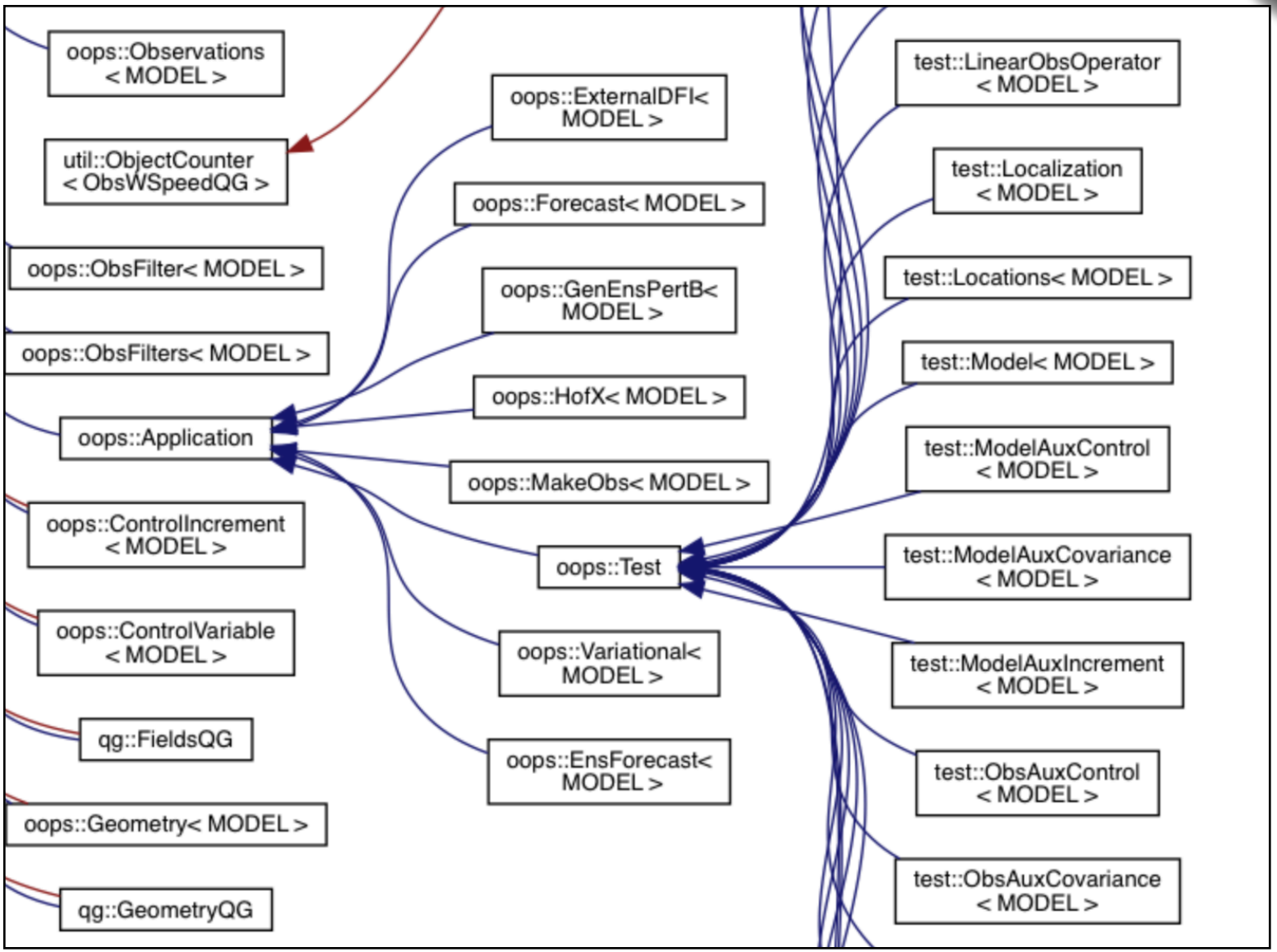
Corresponding code

```
// ------------------------------------------------------------------------------
/*! \brief Interpolation test
 *
 * \details **testStateInterpolation()** tests the interpolation for a given
 * model.  The conceptual steps are as follows:
 * 1. Initialize the JEDI State object based on idealized analytic formulae
 * 2. Interpolate the State variables onto selected "observation" locations
 *    using the getValues() method of the State object.  The result is
 *    placed in a JEDI GeoVaLs object
 * 3. Compute the correct solution by applying the analytic formulae directly
 *    at the observation locations.
 * 4. Assess the accuracy of the interpolation by comparing the interpolated
 *    values from Step 2 with the exact values from Step 3
 *
 * The interpolated state values are compared to the analytic solution for
 * a series of **locations** which includes values optionally specified by the
 * user in the "StateTest" section of the config file in addition to a
 * randomly-generated list of **Nrandom** random locations.  Nrandom is also
 * specified by the user in the "StateTest" section of the config file, as is the
 * (nondimensional) tolerence level (**interp_tolerance**) to be used for the tests.
[…]
```

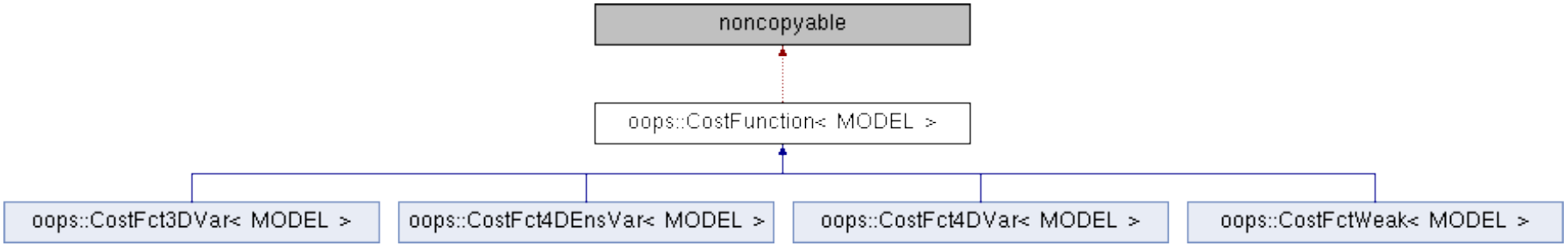
Corresponding code (cont.)

```
[…]
 *
 * This is an equation:
 * \f[ \zeta = \left(\frac{x-x_0}{\lambda}\right)^{2/3} \f]
 *
 * Relevant parameters in the **State* section of the config file include
 *
 * * **norm-gen** Normalization test for the generated State
 * * **interp_tolerance** tolerance for the interpolation test
 *
 * \date April, 2018: M. Miesch (JCSDA) adapted a preliminary version in the
 * feature/interp branch
 *
 * \warning Since this model compares the interpolated state values to an exact analytic
 * solution, it requires that the "analytic_init" option be implemented in the model and
 * selected in the "State.StateGenerate" section of the config file.
 */
```

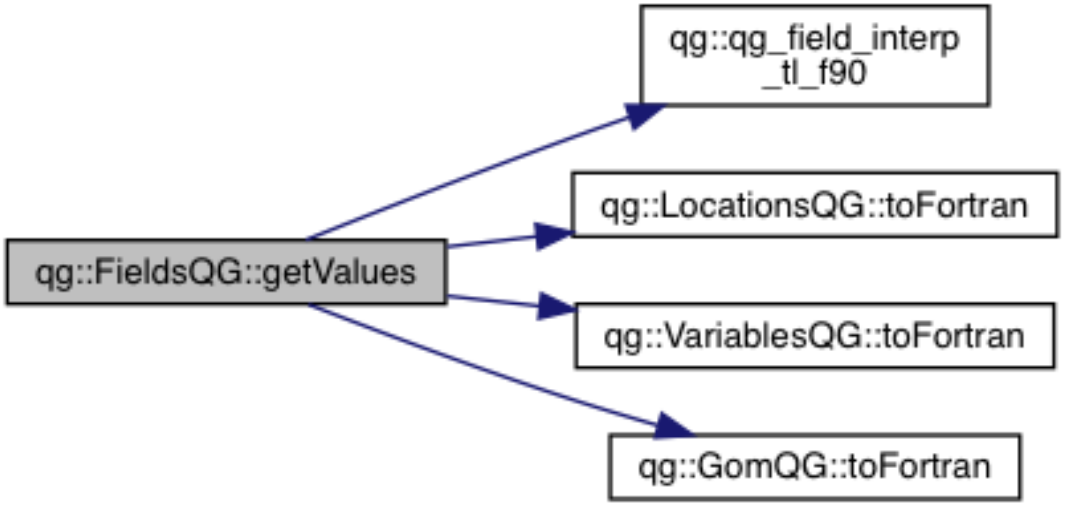
Sample output: class hierarchy



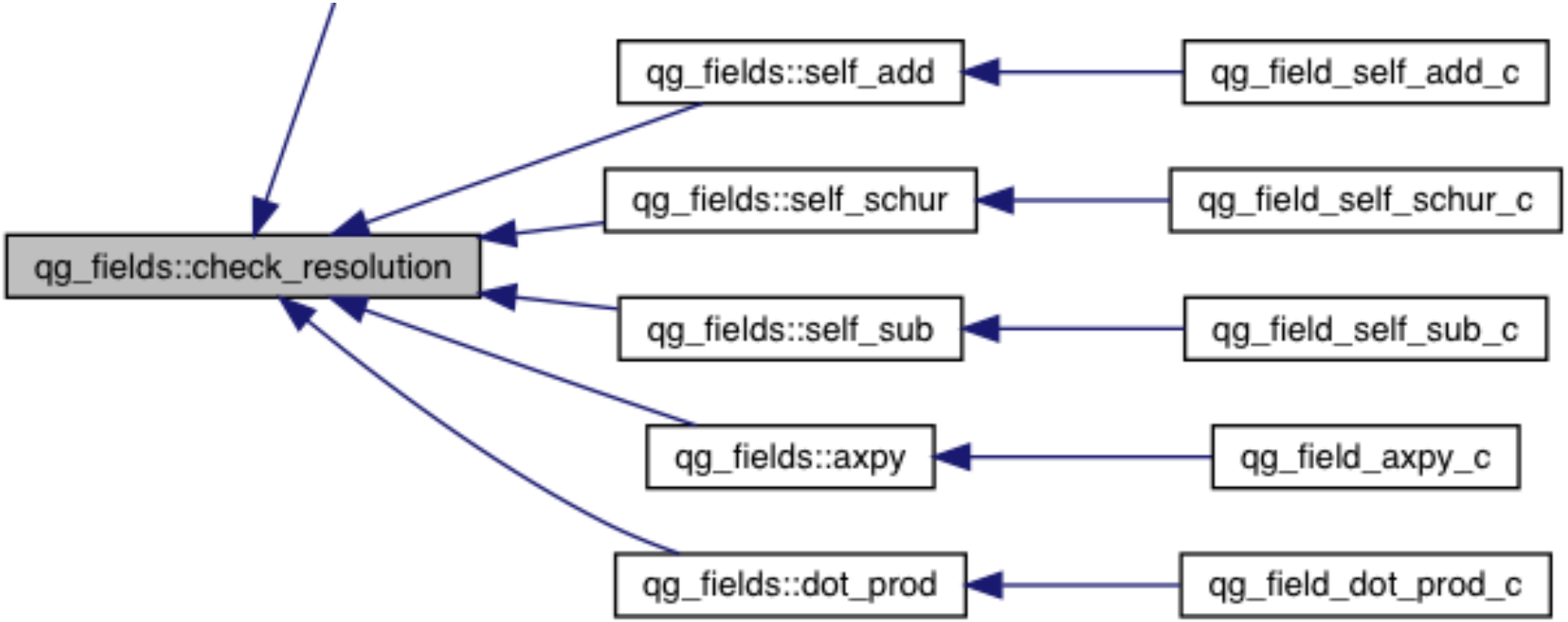
Sample output: inheritance, call graphs



Clickable boxes!

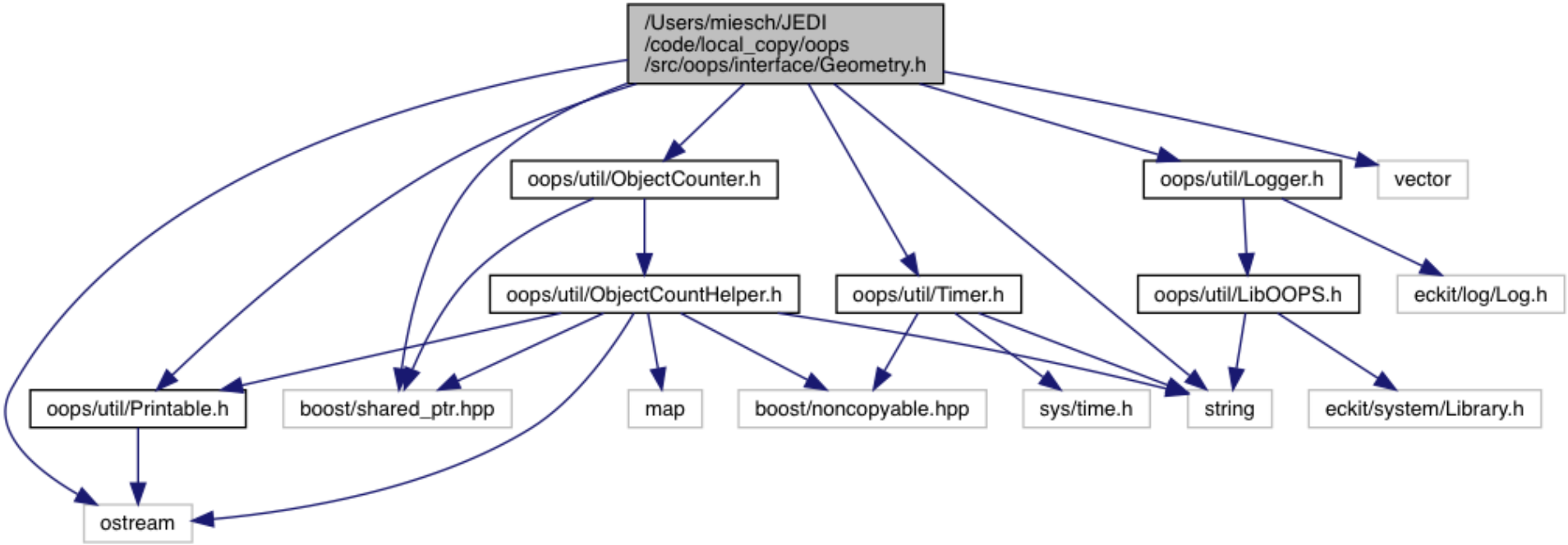


Sample output: caller graphs



Note that these traces end in _c (this is a Fortran routine)
Doxygen has trouble with C++ / Fortran binding
Look for corresponding _f90 routine to follow further

Sample output: include files



Can get complicated!

Other documentation

In a few cases, other sorts of documentation (often pdf) may be available in the Documents directory of a repo

Example: oops

Generally, we plan to link to these pdfs from the Doxygen pages

A Two Level Quasi-geostrophic Model

Mike Fisher, ECMWF

February 8, 2018

1 Introduction

This note describes a simple two-level quasi-gestrophic model, intended for use as a "toy" system with which to conduct idealised studies of data assimilation methods. In developing the model, the emphasis has been placed on speed and convenience rather than accuracy and conservation.

2 The Continuous Equations

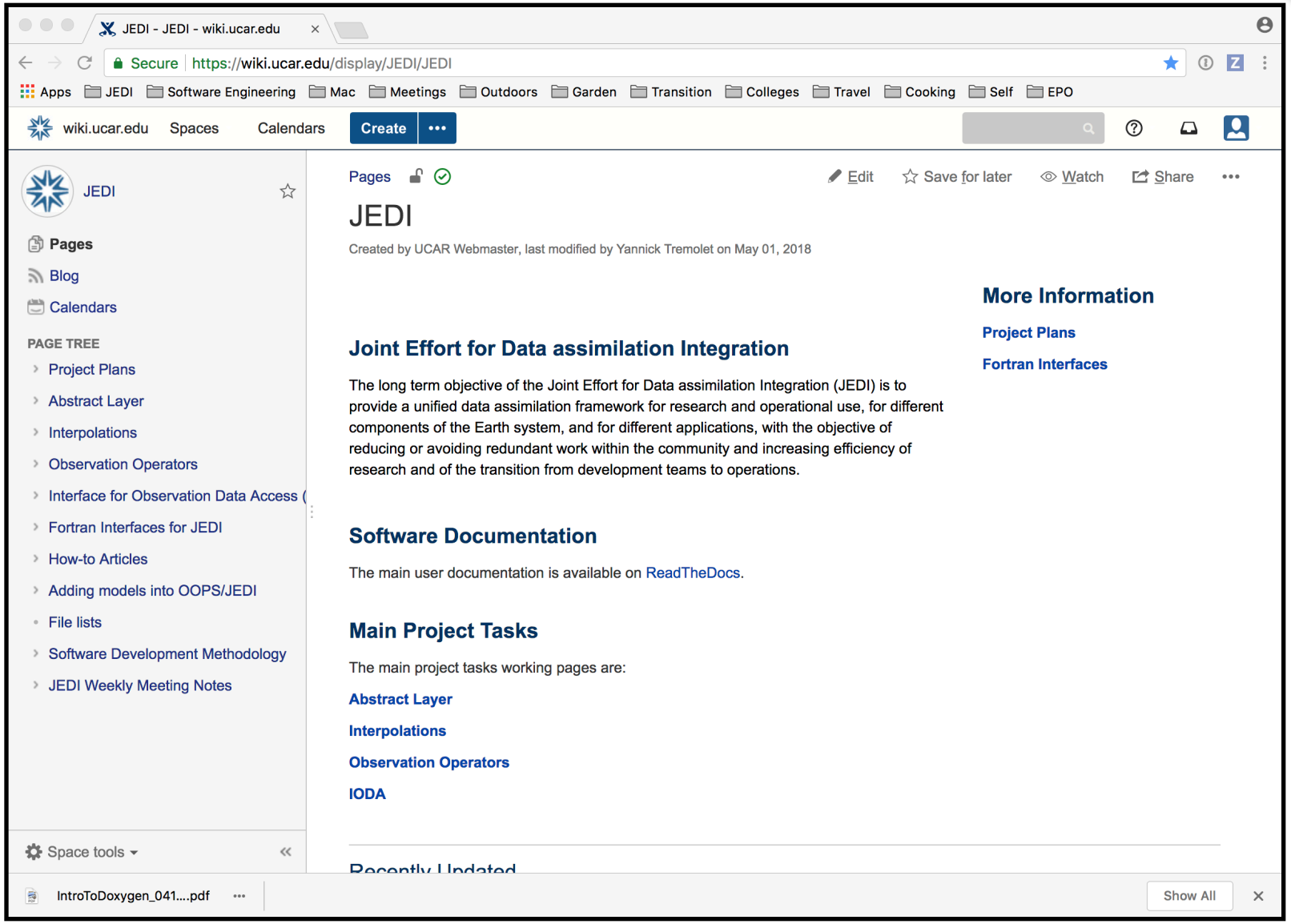
The equations of the two-level model are given by Fandry and Leslie (1984) (see also Pedlosky, 1979 pp386-393), and are expressed in terms of non-dimensionalised variables:

$$\frac{\mathrm{D}q_1}{\mathrm{D}t} = \frac{\mathrm{D}q_2}{\mathrm{D}t} = 0 \tag{1}$$

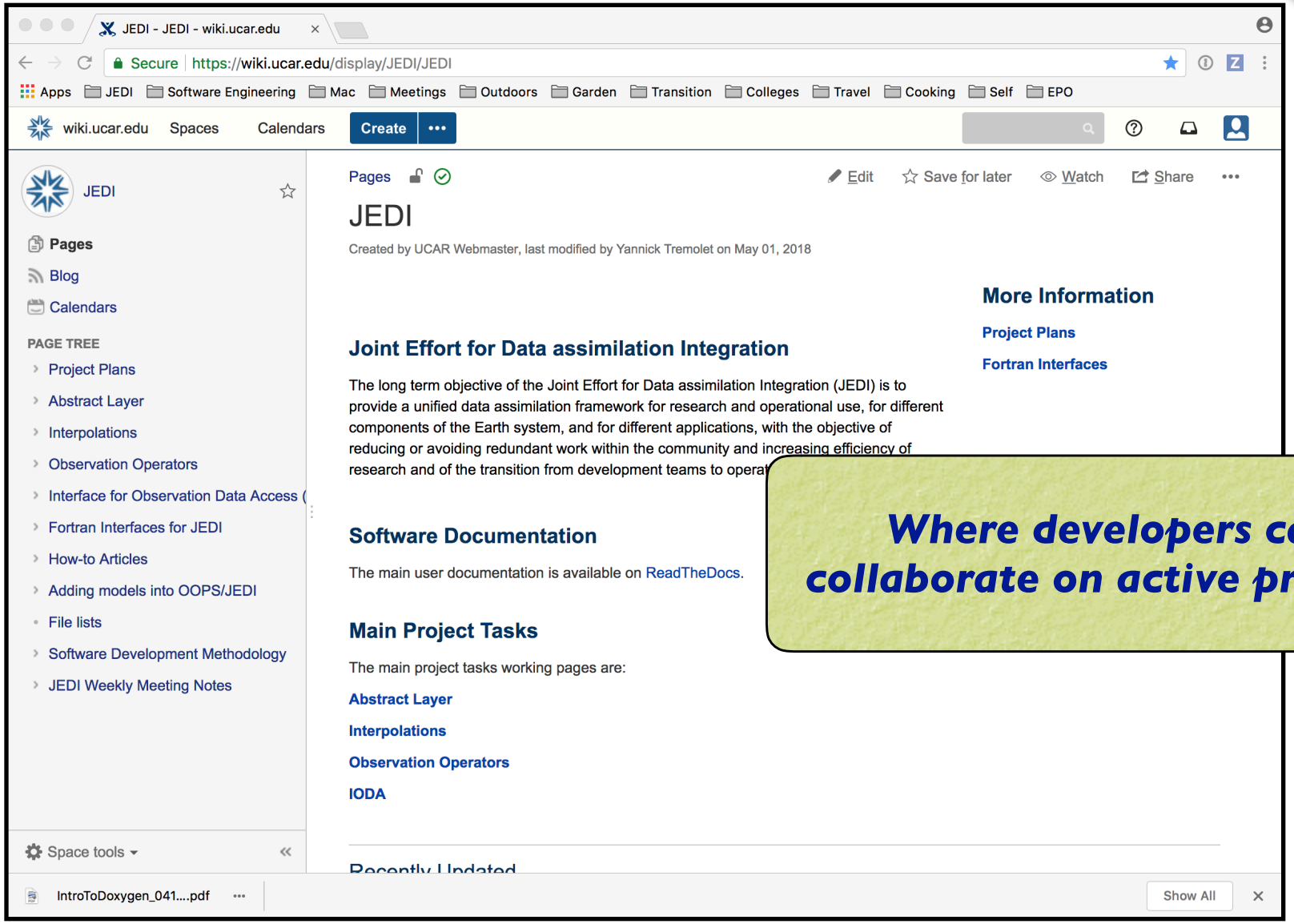
where q_1 and q_2 denote the quasi-geostrophic potential vorticity on each of the two layers, with a subscript 1 denoting the upper layer:

$$q_1 = \nabla^2\psi_1 - F_1(\psi_1 - \psi_2) + \beta y \tag{2}$$
$$q_2 = \nabla^2\psi_2 - F_2(\psi_2 - \psi_1) + \beta y + R_s \tag{3}$$

JEDI Wiki

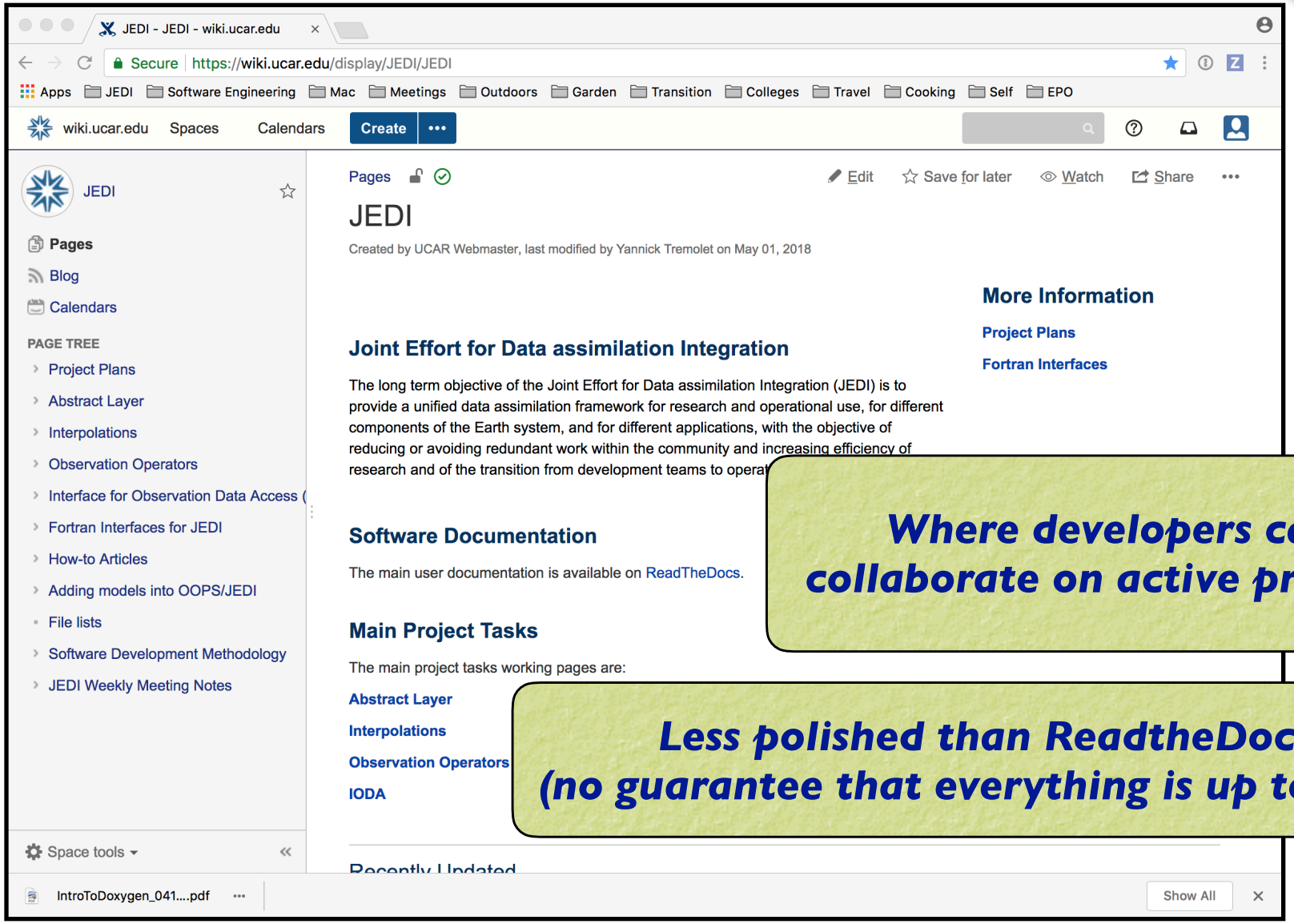

JEDI - JEDI - wiki.ucar.edu
Secure https://wiki.ucar.edu/display/JEDI/JEDI
Apps JEDI Software Engineering Mac Meetings Outdoors Garden Transition Colleges Travel Cooking Self EPO
wiki.ucar.edu Spaces Calendars Create
JEDI
Pages
Blog
Calendars
PAGE TREE
Project Plans
Abstract Layer
Interpolations
Observation Operators
Interface for Observation Data Access (
Fortran Interfaces for JEDI
How-to Articles
Adding models into OOPS/JEDI
File lists
Software Development Methodology
JEDI Weekly Meeting Notes
Space tools
Pages
Edit
Save for later
Watch
Share
JEDI
Created by UCAR Webmaster, last modified by Yannick Tremolet on May 01, 2018
Joint Effort for Data assimilation Integration
The long term objective of the Joint Effort for Data assimilation Integration (JEDI) is to provide a unified data assimilation framework for research and operational use, for different components of the Earth system, and for different applications, with the objective of reducing or avoiding redundant work within the community and increasing efficiency of research and of the transition from development teams to operations.
Software Documentation
The main user documentation is available on ReadTheDocs.
Main Project Tasks
The main project tasks working pages are:
Abstract Layer
Interpolations
Observation Operators
IODA
More Information
Project Plans
Fortran Interfaces
Recently Updated
IntroToDoxygen_041....pdf
Show All

JEDI Wiki



Where developers can collaborate on active projects

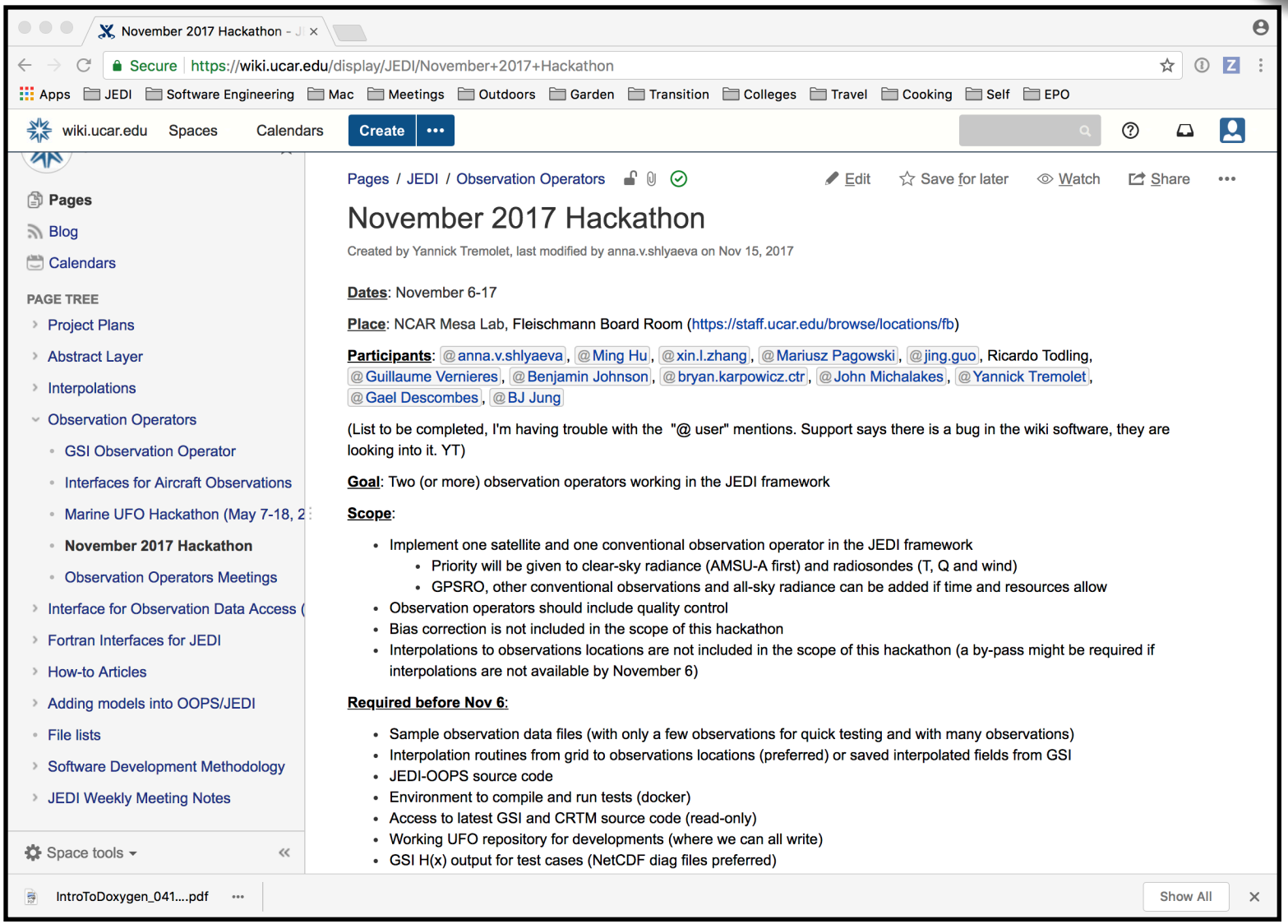
JEDI Wiki



Where developers can collaborate on active projects

Less polished than ReadtheDocs (no guarantee that everything is up to date)

JEDI Wiki


November 2017 Hackathon - J
Secure https://wiki.ucar.edu/display/JEDI/November+2017+Hackathon
Apps JEDI Software Engineering Mac Meetings Outdoors Garden Transition Colleges Travel Cooking Self EPO
wiki.ucar.edu Spaces Calendars Create
Pages
Blog
Calendars
PAGE TREE
Project Plans
Abstract Layer
Interpolations
Observation Operators
GSI Observation Operator
Interfaces for Aircraft Observations
Marine UFO Hackathon (May 7-18, 2
November 2017 Hackathon
Observation Operators Meetings
Interface for Observation Data Access (
Fortran Interfaces for JEDI
How-to Articles
Adding models into OOPS/JEDI
File lists
Software Development Methodology
JEDI Weekly Meeting Notes
Space tools
Pages / JEDI / Observation Operators
Edit
Save for later
Watch
Share
November 2017 Hackathon
Created by Yannick Tremolet, last modified by anna.v.shlyaeva on Nov 15, 2017
Dates: November 6-17
Place: NCAR Mesa Lab, Fleischmann Board Room (https://staff.ucar.edu/browse/locations/fb)
Participants: @anna.v.shlyaeva, @Ming Hu, @xin.l.zhang, @Mariusz Pagowski, @jing.guo, Ricardo Todling, @Guillaume Vernieres, @Benjamin Johnson, @bryan.karpowicz.ctr, @John Michalakes, @Yannick Tremolet, @Gael Descombes, @BJ Jung
(List to be completed, I'm having trouble with the "@ user" mentions. Support says there is a bug in the wiki software, they are looking into it. YT)
Goal: Two (or more) observation operators working in the JEDI framework
Scope:
Implement one satellite and one conventional observation operator in the JEDI framework
Priority will be given to clear-sky radiance (AMSU-A first) and radiosondes (T, Q and wind)
GPSRO, other conventional observations and all-sky radiance can be added if time and resources allow
Observation operators should include quality control
Bias correction is not included in the scope of this hackathon
Interpolations to observations locations are not included in the scope of this hackathon (a by-pass might be required if interpolations are not available by November 6)
Required before Nov 6:
Sample observation data files (with only a few observations for quick testing and with many observations)
Interpolation routines from grid to observations locations (preferred) or saved interpolated fields from GSI
JEDI-OOPS source code
Environment to compile and run tests (docker)
Access to latest GSI and CRTM source code (read-only)
Working UFO repository for developments (where we can all write)
GSI H(x) output for test cases (NetCDF diag files preferred)
IntroToDoxygen_041....pdf
Show All

JEDI Wiki: Weekly Meeting Notes

May 3, 2018 - JEDI - wiki.ucar.

Secure | https://wiki.ucar.edu/display/JEDI/May+3%2C+2018

Apps | JEDI | Software Engineering | Mac | Meetings | Outdoors | Garden | Transition | Colleges | Travel | Cooking | Self | EPO

wiki.ucar.edu Spaces Calendars Create

Blog
Calendars

PAGE TREE
- Project Plans
- Abstract Layer
- Interpolations
- Observation Operators
- Interface for Observation Data Access (
- Fortran Interfaces for JEDI
- How-to Articles
- Adding models into OOPS/JEDI
- File lists
- Software Development Methodology
- JEDI Weekly Meeting Notes
  - April 12, 2018
  - April 19, 2018
  - April 26, 2018
  - **May 3, 2018**
  - May 10, 2018
  - May 17, 2018
  - May 24, 2018

Space tools

Pages / JEDI / JEDI Weekly Meeting Notes Edit Save for later Watching Share

May 3, 2018

Created by Mark Miesch, last modified by Stephen Herbener on May 03, 2018

Most of today's meeting was concerned with the reorganization of the ObsSpace classes in oops, ufo, and ioda.

Xin started the discussion by pointing out multiple places in ufo where code is duplicated, both in terms of the file structure and the code itself. He then went on to illustrate several examples of conditional execution based on if/else if statements. This could be cleaned up substantially and optimized with a more object-oriented approach.

For this reason, there is an effort at JCSDA (led by Xin, Steve, and Yannick) to reorganize the ObsSpace data structure in order to:

- Reduce duplicated subroutines
- Simplify the APIs
- Re-design ObsSpace data structure

Then Steve shared similar concerns and efforts, focusing in particular on the reading and writing of data in ioda:

IntroToDoxygen_041....pdf

Show All

Doxygen Resources

JEDI Doxygen page

https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/en/latest/developer/developer_tools/doxygen.html

Doxygen Users Manual

http://www.stack.nl/~dimitri/doxygen/manual/index.html

Installation? Already installed in the JEDI singularity container

Binaries available for download on:

http://www.stack.nl/~dimitri/doxygen/download.html

Or, on a Mac:

brew install doxygen