EFDA

EUROPEAN FUSION DEVELOPMENT AGREEMENT

Task Force
INTEGRATED TOKAMAK MODELLING

15/09/2010

IMP5: Quick introduction to documentation with Doxygen

T. Johnson

Outline

- For a code to be fully integrated into the ITM there has to be two types of documentation:
  - For the developer
  - For the user
- Documentation can be written in word/LaTex…
  - Documentation for developer: an effective choise is to use *documentation tools*, i.e. tools that automatically generate documentation directly from your Fortran/C/IDL/Java/… source files.
  - Doxygen – commonly used within ITM
  - Fortdocu – never tried myself
  - Javadoc – good for Java
- Suggestion: If you ever plan to use Doxygen – start coding with Doxygen in mind already today!
- I'll show quick introduction to Doxygen and a few examples

Doxygen: step 1

- You'll need Doxygen installed on your computer
  - Unfortunatelly Doxygen at the gateway is not working as it should; something wrong with the Fortran setting I understand.
  - For Mac users, Doxygen can be installed with Darwinports (Macports) http://darwinports.com
  - Everythings freeware!
  - …and theres lots of documentation/tutorials online
    - I hope this will give you glimps of what Doxygen can do and maybe help you get you started – you can generate test-documentation for your code in 5 min!

Doxygen: getting started

Once installed, you have to setup Doxygen for your project

•Go to directory with your source files

•Generate a default configuration file:

doxygen –g <filename>

example of auto-generated configuration file

•Now run

doxygen <filename>

Still no documentation...

- You'll get HTML/LaTex files (in ./html/ and ./latex/)

•It can look much better! So next; configure doxygen for your project

```
# Doxyfile 1.7.1

# This file describes the settings to be used by the documentation system
# doxygen (www.doxygen.org) for a project
#
# All text after a hash (#) is considered a comment and will be ignored
# The format is:
#       TAG = value [value, ...]
# For lists items can also be appended using:
#       TAG += value [value, ...]
# Values that contain spaces should be placed between quotes (" ")

#---------------------------------------------------------------------------
# Project related configuration options
#---------------------------------------------------------------------------

# This tag specifies the encoding used for all characters in the config file
# that follow. The default is UTF-8 which is also the encoding used for all
# text before the first occurrence of this tag. Doxygen uses libiconv (or the
# iconv built into libc) for the transcoding. See
# http://www.gnu.org/software/libiconv for the list of possible encodings.

DOXYFILE_ENCODING      = UTF-8

# The PROJECT_NAME tag is a single word (or a sequence of words surrounded
# by quotes) that should identify the project.

PROJECT_NAME           =

# The PROJECT_NUMBER tag can be used to enter a project or revision number.
# This could be handy for archiving the generated documentation or
# if some version control system is used.

PROJECT_NUMBER         =

# The OUTPUT_DIRECTORY tag is used to specify the (relative or absolute)
# base path where the generated documentation will be put.
# If a relative path is entered, it will be relative to the location
# where doxygen was started. If left blank the current directory will be used.

OUTPUT_DIRECTORY       =
...
```

Doxygen: configuration

- Configuration can be done by hand or in doxwizard (I've never tried it)
- Some changes to try out:
  - PROJECT_NAME = <my_projects_name>
  - OUTPUT_DIRECTORY = docs/
    (this will put all documentation in directory docs/ – very handy!)
  - FILE_PATTERNS= *.f90 *.F *.f
    (not necessary; default settings seem ok)
  - OPTIMIZE_FOR_FORTRAN = YES
    (or OPTIMIZE_FOR_C = YES)
  - EXTRACT_ALL = YES
  - SOURCE_BROWSER = YES
  - GENERATE_LATEX = NO
    (in case you dont need the LaTex output; Latex generates *.tex pages and one huge pdf file.)

For call/caller graphs

- HAVE_DOT = YES
- CALL_GRAPH = YES
- CALLER_GRAPH = YES

Example from ASCOT

subroutine ripple_module::VesWidthAtZ (REAL(kind=params_wp),intent(in) ZVAL,
REAL(kind=params_wp),intent(out) RLEFT,
REAL(kind=params_wp),intent(out) RRIGHT,
INTEGER,intent(out) I1,
INTEGER,intent(out) I2
)

Definition at line 1131 of file ripple_module.f90.

References channel::channel_out, magnrz::magnrz_linr, mag…

Referenced by ripple_init().

Here is the caller graph for this function:

ripple_module::VesWidthAtZ

subroutine ripple_module::ripple_dealloc ( )

Definition at line 1195 of file ripple_module.f90.

References ripple_tbr1, ripple_tbrbx1, ripple_tbrbx2, ripple_tbz1, ripple_tdbrx2, ripple_tdbz1, ripple_tdbz1r, ripple_tdbz1z, ripple_tdrbx2r, and ripple_tdrbx2z.

Variable Documentation

INTEGER,save ripple_module::ripple_ncoil

Definition at line 78 of file ripple_module.f90.

Referenced by ASC000(), ASCBKG(), ASCCVG(), ASCCXD(), ASCEQU(), ASCIN1(), ASCINP(), modAscipt::ASCIPT(), ASCVCS(), ASCWVG(), BTPWVG(), fullorbit_module::fullorbit_FOtoGC(), bkg_generator::init_mgfield(), InputOptionsCreate(), and andiff::moveInRhoKeepVelocity().

INTEGER,save ripple_module::ripple_nc1

Definition at line 78 of file ripple_module.f90.

Referenced by ASCINP(), and ripple_eval().

ripple_module.f90

Go to the documentation of this file.

```
00001 MODULE ripple_module
00002
00003   USE params, ONLY: params_ir, params_iz, params_wp
00004   IMPLICIT NONE
00005 !
00006 !-----------------------------------------------------------------------
00007 !
00008 ! Magnetic ripple variables:
00009
00010 ! ripple_b      - ripple magnetic field magnitude (T)
00011 ! ripple_bt     - toroidal component of ripple magnetic field (T)
00012 ! ripple_bp     - poloidal component of ripple magnetic field (T)
00013 ! ripple_br     - R component of ripple magnetic field (T)
00014 ! ripple_bz     - z component of ripple magnetic field (T)
```

Doxygen: Documentation in the source code

There's still no documentation in there!
But it is all for free - and when combining it with written documentation, that's what's useful!

So how do I make the documentation in my code appear in the Doxygen documents?

In Fortran: use "!>", e.g.

"!> This text will appear in Doxygen documents – wohoo :)"

In C / C++/Java:

```
"/**  Here's a doxygen
   *   section for a C-code!
   */"
```

For fortran code: in Emacs, run "Replace regexp…" from "!" to "!>"
(If you use Emacs, but don't know about "Replace regexp…"; let me know and I'll show you)
NOTE: if you start writing comments with "!>" it will be easy to use Doxygen in future.

Special commands

There are a number of FLAGs that you can put in to your source code, which the doxygen compiler will identify, e.g.:

- `/mainpage` the following text will appear as the Main-page/ front-page of your Doxygen documentation.
- `/version` <No.> allow you to define the version number in your source code rather then in the Doxyfile.
- `/author` <name>
- `/param` <abc> describes the parameter <abc> of an function
- `/return` describes the return value of an function
- `/date` <date>
- `/note` Will add a note keyword in bold if desired.
- `/brief`a short code description
- `/file` <abc.de> describes the file <abc.de>
- `/todo` <task> adds description of what needs doing to a to-do list.
- `/bug` <description> Adds <destription> to a list of bugs

Doxygen example (from RFOF)

Latex Equation

```
!-------------------------------------------------------------------------------
!>
!> function quasilinear_RF_diffusion_coeff(marker,RFlocal,mem) result(diffusion)
!>
!> Quasilinear diffusion coefficient for resonant interactions between
!> RF wave field and charged particle in an inhomogeous magnetic field.
!> This is a local operator given by the local magnetic fields and wave
!> quantities and their time derivatives in the frame of the particles.
!>
!> The diffusion coefficient in \f$ I_\perp \f$ is:
!> \f[ D = \frac{1}{2} \left| \tau Ze v_\perp E_{eff} \right|^2 \f]
!> See [ L.-G. Eriksson, Nuclear Fusion, 2005 ]
!>
!> \attention \c dIperp is in units [MeV].
!>
!> INPUT
!> \param marker    Properties of the marker (weight, mass, charge, velocity...)
!> \param RFlocal   Local properties of the RF wave field (\f$|B|, RB_\phi,\dots\f$)
!> \param mem       Short term memory of the resonance condition along the orbit
!>
!> OUTPUT
!> \param diffusion (out) Quasilinear scattering coefficient :
!>                        \f$ D = <dI_\perp dI_\perp> \f$
!>                        during one crossing of the resonance.
!>
!-------------------------------------------------------------------------------
function quasilinear_RF_diffusion_coeff(marker,RFlocal,mem) result(diffusion)

 ! Input
 type(particle), intent(in) :: marker
 type(rf_wave_local), intent(in) :: RFlocal
 type(resonance_memory), intent(in) :: mem
```

real(8) RFOF_kick::quasilinear_RF_diffusion_coeff (type(particle),intent(in) marker,
type(rf_wave_local),intent(in) RFlocal,
type(resonance_memory),intent(in) mem
)

function quasilinear_RF_diffusion_coeff(marker,RFlocal,mem) result(diffusion)

Quasilinear diffusion coefficient for resonant interactions between RF wave field and charged particle in an inhomogeous magnetic field. This is a local operator given by the local magnetic fields and wave quantities and their time derivatives in the frame of the particles.

The diffusion coefficient in $I_\perp$ is:

$$D = \frac{1}{2} |\tau Z e v_\perp E_{eff}|^2$$

See [L.-G. Eriksson, Nuclear Fusion, 2005]

Attention:
`dIperp` is in units [MeV].

INPUT

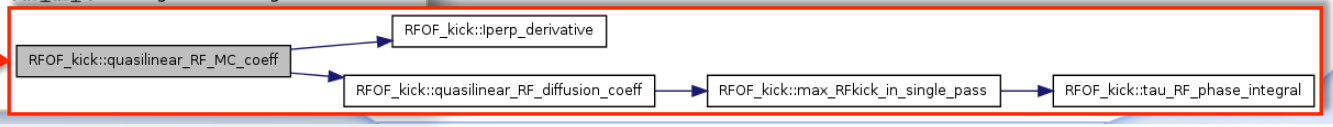
Parameters:
marker Properties of the marker (weight, mass, charge, velocity...)
RFlocal Local properties of the RF wave field ($|B|, RB_\phi, \dots$)
mem Short term memory of the resonance condition along the orbit

OUTPUT

Parameters:
diffusion (out) Quasilinear scattering coefficient : $D = < dI_\perp dI_\perp >$ during one crossing of the resonance.

Here is the call graph for this function:

Here is the caller graph for this function:



`\param` help Doxygen identify the description of your parameters

Doxygen example: main page

Generate the ”main-page” (your index.html) by the label `\mainpage`, e.g.

- in Fortran code, or
- in non-fortran file <test>.dox

```
/*!
\mainpage RFOF - Radio Frequency Monte
Carlo Library for Orbit Following Codes

\author Thomas Johnson, Antti Salmi\n
Developed for the EU-ITM, Integrated
modelling project 5, ACT4-2010.

\n<hr>\n

<h2>
Code status:
</h2>

- The code can find
position for non-ac
i.e. judge if parti
and predict the tim
future and past resonance. For time-
acceleration the time derivatives are has
to be treated over orbit time rather then
simulation time and special care has to
be taken when the rate of time-
acceleration is changed.
. . .
```

Doxygen handles most HTML functions

How to couple your orbit following code to RFOF

1. Generate the data structure needed in RFOF. These are NOT global and your code has to keep track of them (keep your pointers until data is deallcocated). RFOF provides constructors for these data structures:
   - **THE MARKERS:** The marker structure are set in the subroutine `set_marker_pointers` in the module RFOF_markers. The markers data, e.g. v-parallel and psi, are all stored as pointer to your internal markers; i.e. both your code RFOF access the same memory and changes set in RFOF are immediately seen in your code, ad vice versa. This routine therefore require you to provide TARGETs for all marker variables. It also means that this structure is set only once per marker.
   - **THE RESONANCE MEMORY:** In order to evaluate the wave-particle interaction strength you need to know the time derivative of the cyclotron frequency of the marker, thus the short term time-history of the particle needs to be stored. This is the perpose of the "resonance-memory". The memory is contructed by the subroutine `constructor_rf_resonance_memory_matrix` in the module RFOF_resonance_memory. Note that this memory is specific for each particle and has to be reset every time you switch to tracing a new particle. To reset the memory use the subroutine `reset_rf_resonance_memory_matrix` in the module RFOF_resonance_memory.
   - THE RF WAVE FIELD: The RF wave field is handled entirely inside RFOF. The contructor is `dummy_rf_wave_field` in the module RFOF_resonance_memory. At the moment this constructor generates a dummy field with constant components over the entire plasma.
   - THE RF-INTERACTION DIAGNOSTICS: The diagnostics is handled entirely inside RFOF. The contructor is `contructor_RFOF_cumlative_diagnostics` in the module RFOF_diagnostics.
2. Write a subroutine to provide RFOF with your magnetic field data. The subroutine should be called local_magnetic_field_interface_to_RFOF and the interface for this routine is given below.
3. Now we are ready to give the markers RF "kicks". The subroutine to call from your orbit code is `RFOF_master` in the RFOF_main module; this routine should be called after every time step performed by your orbit solver. The routine will first check is the particle is in resonance with any of the wave-modes in your RF wave field (resonance here mean "close enough to the resonance"). If it is in resonance then it will give the particle RF "kicks" for each resonant wave mode. If the particle is not in resonance it will check if the particle has crossed any resonance without recieving a kick. If so, an error flag is raised, and an estimate for the time at which the particle crossed the resonance is returned. It is suggest the you refine and redo your time step such that the new time step ends at the estimated time of the resonance. In case the step is successful, then the `RFOF_master` will return the time at which the next resonance should occure. You may then use this time to decide the length of the folloing time step.
4. Every time you switch from tracing one marker to the next you are have to both set the marker pointers as in 1.1 and to reset the resonance memory as described in 1.2.
5. In later versions the strength of the RF wave field should be updated regularly. This is not implemented yet.
6. At the end of the program all allocated memory should be deallocated. The destructors you need are:
   - `rf_wave_destructor_matrix` from the module RFglobal.
   - `destructor_rf_resonance_memory_matrix` from the module resonanceMemory.
   - `detructor_RFOF_cumlative_diagnostics` from the diagnostics module.