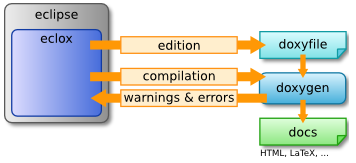
**Eclipse Install Doxygen and Graphviz**

Mingo Hua, Jerhua.

根据下面的网页修改而来，但重点介绍Doxgen和Graphviz的安装和使用。 <https://mcuoneclipse.com/2012/06/25/5-best-eclipse-plugins-1-eclox-with-doxygen-graphviz-and-mscgen/>

**Eclox+Doxygen+Graphviz+Mscgen**. Yes, it is a single Eclipse plugin (Eclox) for Doxygen, and with two other powerful tools.  It solves a typical engineering problem: “How to document my project? And how to keep it up-to-date?”.

Like many other engineers, I do not like to write documentation. Because it is painful. I want to write code and program. Writing documentation for it should be fun too. And it should solve the problem that the documentation does not match what has been implemented. I’m a big fan of the ‘single source’ concept: information has to be in a single place, and not copied and distributed among different places. And here my #1 helps me doing this.

[](https://mcuoneclipse.files.wordpress.com/2012/06/eclox-and-eclipse.png)

eclox and eclipse

In a traditional way the following flow is used:

1. Create a design, specify and document the API
2. Implement the software
3. Write the user documentation
4. Ship it
5. Maintain and ship; and again, and again, and …

The problem starts with the fact, that rarely the implementation matches the initial design. It even gets worse after a few maintenance cycles: it is very hard to keep the documentation in sync with the actual implementation and project sources. And the user documentation will be easily out of sync too.

It is already hard to write good documented code, and writing good source comments is an art on its own. Why not using the my precious source documentation in the sources and use it for the ‘written’ documentation? This is where I use **Doxygen**, **GraphViz**, **Mscgen** and the Eclipse **Eclox** plugin, all of them open source.

**Doxygen** is a compiler which generates documentation out of source files. **Graphviz** is a package to draw diagrams and graphs. **Mscgen** is similar to Graphviz, but simpler and optimized for message sequence diagrams. And **Eclox** is an Eclipse plugin which integrates everything into Eclipse.

**Installation**

Here are the links to download the needed tools:

1. **Doxygen**: [http://www.stack.nl/~dimitri/doxygen/](http://www.stack.nl/%7Edimitri/doxygen/) or [www.doxygen.org](http://www.doxygen.org)
2. **Graphviz**: <http://www.graphviz.org>
3. **Mscgen**: <http://www.mcternan.me.uk/mscgen/>

**NOTE: As of Feb 2018, the doxygen Eclox project has been moved to the following site:** [**https://anb0s.github.io/eclox/**](https://anb0s.github.io/eclox/)**. I recommend that you download the zip file from that site.**

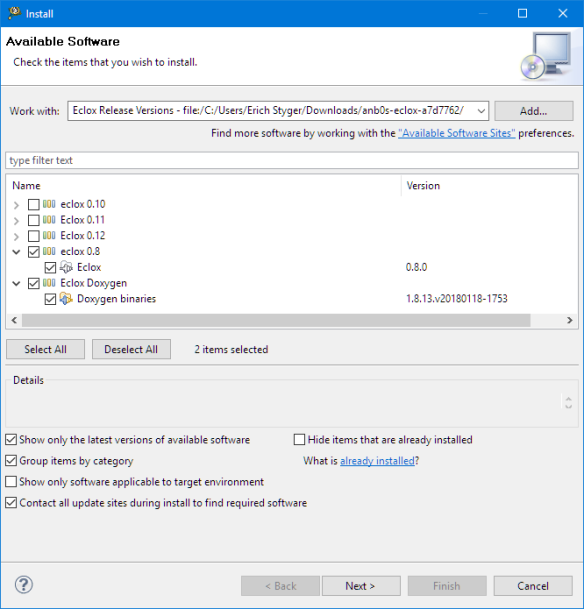
**For CodeWarrior for MCU I’m using the 0.8 version and that one works well. As CW for MCU 11.0 and earlier use an older Eclipse version, do \*not\* upgrade doxygen to a later version: only use 0.8 version.**

**For others, can choose the higher Eclox version.**

Doxygen is required, Graphviz highly recommended, and Mscgen is optional.

Download the zip file and extract it. In Eclipse, use the menu Help > Install New Software and point to the directory where you have extracted the file (or input the link:[**https://anb0s.github.io/eclox/**](https://anb0s.github.io/eclox/)).

If using CodeWarrior for MCU 10.x or 11.x (Eclipse 4.2 Juno based), then only use the 0.8 version:

[](https://mcuoneclipse.files.wordpress.com/2012/06/installing-eclox.png)

Installing Eclox



安装完成之后，重启eclipse，会看到上图的@图标。

**Doxygen**

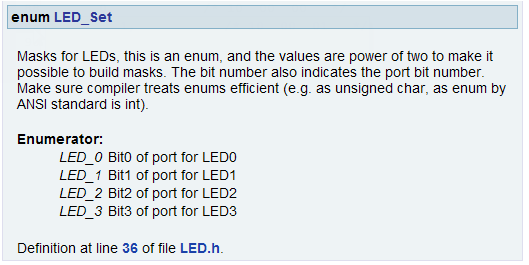
With Doxygen my project sources **\*are\*** the documentation. Doxygen is using the comments in the code to generate the documentation. It is a compiler which compiles source (or text) files and extracts the embedded information. It supports many programming languages (C, C++, Java, …), and different output formats: HTML and PDF are very popular ones.

Doxygen is able to generate documentation from standard sources. But there are multiple ways of using special commenting styles to extend the information created. The Doxygen documentation lists [different commenting styles](http://www.stack.nl/%7Edimitri/doxygen/docblocks.html).

With this, the following source is compiled by Doxygen:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*! Masks for LEDs, this is an enum, and the values are power of two to make it       possible to build masks. The bit number also indicates the port bit number.        Make sure compiler treats enum's efficient (e.g. as unsigned char, as enum        by ANSI standard is int). \*/    typedef enum {      LED\_0 = (1<<0), /\*< Bit0 of port for LED0 \*/      LED\_1 = (1<<1), /\*< Bit1 of port for LED1 \*/      LED\_2 = (1<<2), /\*< Bit2 of port for LED2 \*/      LED\_3 = (1<<3)  /\*< Bit3 of port for LED3 \*/  } LED\_Set; |

It produces the following HTML output documentation:

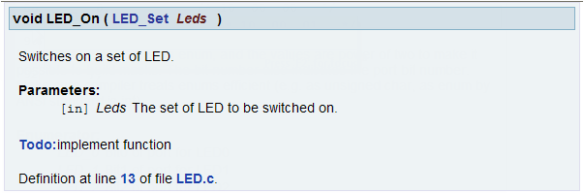
[](https://mcuoneclipse.files.wordpress.com/2012/06/enum-led.png)

enum LED\_Set HTML Documentation

Another example is to document functions with their parameters:

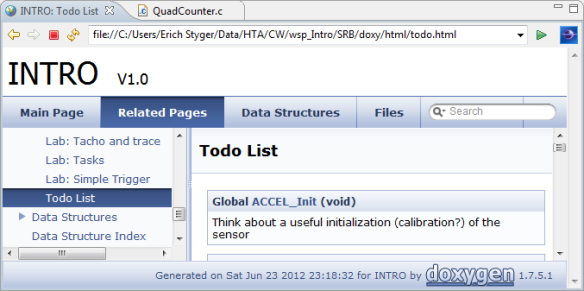
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\*\*   \* \brief Switches on a set of LED.   \* \param[in] Leds The set of LED to be switched on.   \*/  void LED\_On(LED\_Set Leds);    void LED\_On(LED\_Set Leds) {    /\*! \todo implement function \*/  } |

The corresponding HTML documentation will look like this:

[](https://mcuoneclipse.files.wordpress.com/2012/06/led_on-documentation.png)

LED\_On() Documentation

Note the **\todo** with will automatically added to a ‘To Do’ list in the documentation:

[](https://mcuoneclipse.files.wordpress.com/2012/06/todolist.png)

Todo List

**Graphviz**

Graphviz的配置，在原网页博客上没有说明，增加如下：

在网站<https://graphviz.gitlab.io/_pages/Download/Download_windows.html>下载安装包。本人测试时下载了[graphviz-2.38.zip](https://graphviz.gitlab.io/_pages/Download/windows/graphviz-2.38.zip) 文件，不用管理员权限。

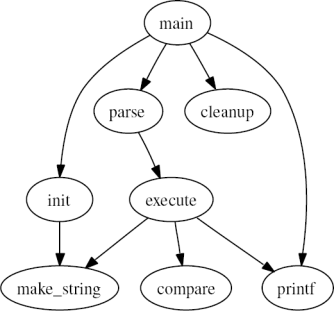
将下载的文件解压到一个路径下，例如C:\graphviz-2.38。



[Graphviz](http://www.graphviz.org) is open source software to create graphs using the *dot* language. Dot code can be used with Doxygen. With the *dot* language I define nodes and edges, and the tool will place them automatically. For example I can define a graph with following *dot* code (example from the *dot* guide):

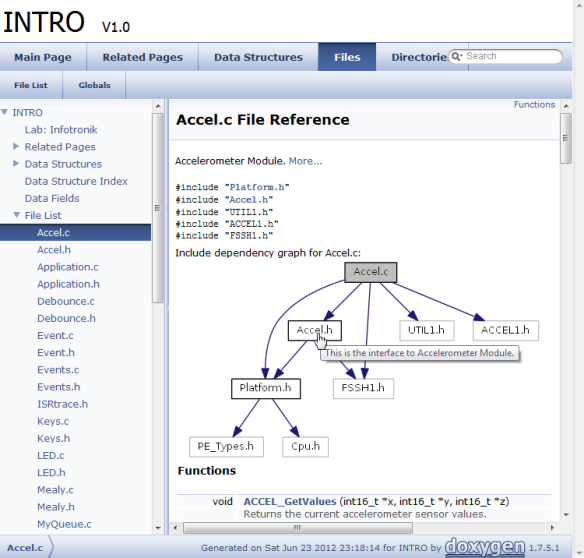
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\*\*  \dot  digraph G {  main -> parse -> execute;  main -> init;  main -> cleanup;  execute -> make\_string;  execute -> printf  init -> make\_string;  main -> printf;  execute -> compare;  }  \enddot  \*/ |

This then will create the following graph (source: dot guide):

[](https://mcuoneclipse.files.wordpress.com/2012/06/small-dot-graph.png)

small dot graph

The *dot* language makes it easy to create machine generated graphs too. That way *dot* is used in many other tools as visualization tool. And Doxygen is using of *dot* to create dependency graphs:

[](https://mcuoneclipse.files.wordpress.com/2012/06/dependency-graph.png)

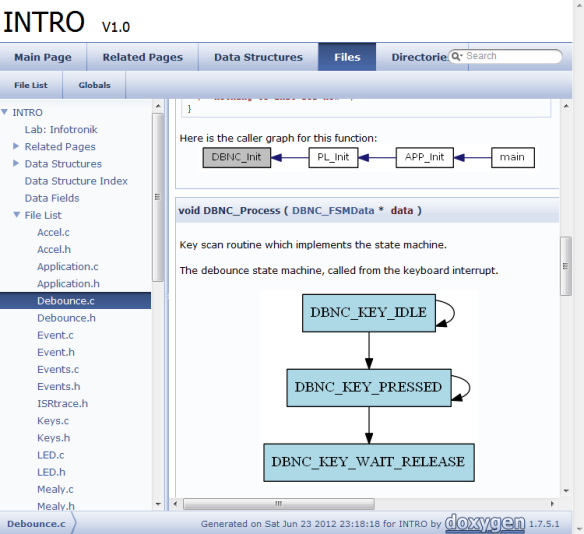
Dependency Graph

A nice feature is as well that the nodes have hyperlinks: click on a node an it jumps to the source file :-).

I’m using dot in my sources to enrich the documentation with graphs (a pictures may say more than 1000 words). Below is an example Doxygen/dot code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\*! \brief Key scan routine which implements the state machine.  \dot  digraph example\_api\_graph {  node [shape=box];  DBNC\_KEY\_IDLE    [fillcolor=lightblue,style=filled,label="DBNC\_KEY\_IDLE" ];  DBNC\_KEY\_PRESSED [fillcolor=lightblue,style=filled,label="DBNC\_KEY\_PRESSED" ];  DBNC\_KEY\_RELEASE [fillcolor=lightblue,style=filled,label="DBNC\_KEY\_WAIT\_RELEASE"];  DBNC\_KEY\_IDLE -> DBNC\_KEY\_PRESSED -> DBNC\_KEY\_RELEASE ;  DBNC\_KEY\_PRESSED -> DBNC\_KEY\_PRESSED ;  DBNC\_KEY\_IDLE -> DBNC\_KEY\_IDLE ;  }  \enddot  \*/ |

This will generate the following output:

[](https://mcuoneclipse.files.wordpress.com/2012/06/debounce-graph.png)

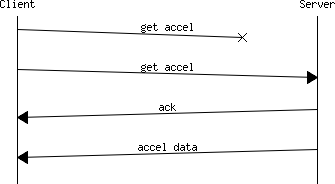
Debounce graph

**Mscgen**

Mscgen is similar to Graphviz, but specialized for **Message Sequence Charts** (hence the Msc in its name). With Mscgen I can add following doxygen/msc code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*\*  \msc  arcgradient = 8;  a [label="Client"],b [label="Server"];  a-xb [label="get accel"];  a=>b [label="get accel"];  a<=b [label="ack"];  a<=b [label="accel data"];  \endmsc  \*/ |

This will create following message sequence diagram into my documentation:

[](https://mcuoneclipse.files.wordpress.com/2012/06/mscgen-example.png)

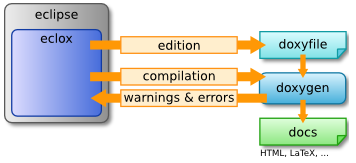
Mscgen Example Chart

Mscgen is ideal for any communication flow diagrams and greatly simplify and enhance my documentation.

**Eclox**

All of above can be used without any IDE. But it is a lot of more fun within the Eclipse framework. And here is where [**Eclox**](http://home.gna.org/eclox/#overview) comes into play: Eclox is a plugin which integrates Doxygen (and all above tools) into Eclipse and makes it really easy to use:

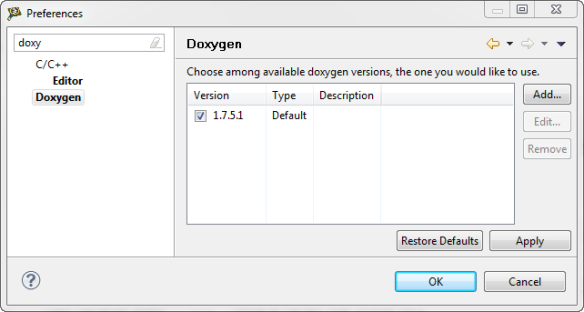
* Seamless build integration
* Editor coloring for Doxygen comments
* User interface for Doxygen configuration files
* Editor integration with automatic commenting
* Warning/Error parser
* Wizard to create configuration files

[](https://mcuoneclipse.files.wordpress.com/2012/06/eclox-and-eclipse.png)

Eclox and Eclipse

**Eclox Plugin**

Eclox automatically detects an existing Doxygen installation and reports it in the Preferences:

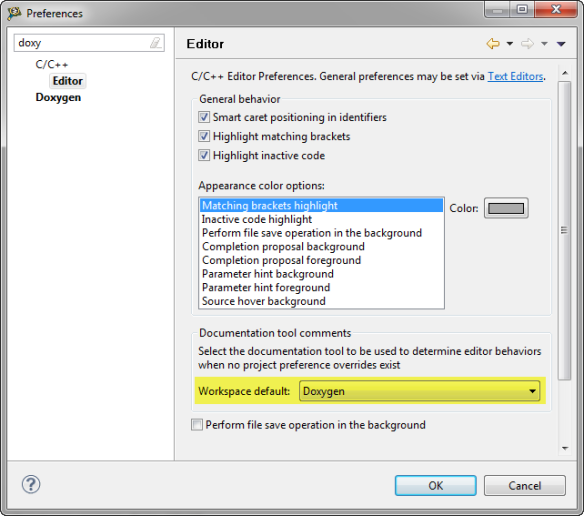
[](https://mcuoneclipse.files.wordpress.com/2012/06/doxygen-version.png)

Doxygen Version

In case this fails, then this [link](http://stackoverflow.com/questions/2108586/how-to-use-eclox-the-doxygen-plugin-for-eclipse) gives some guidance how to set the settings in a manual way.

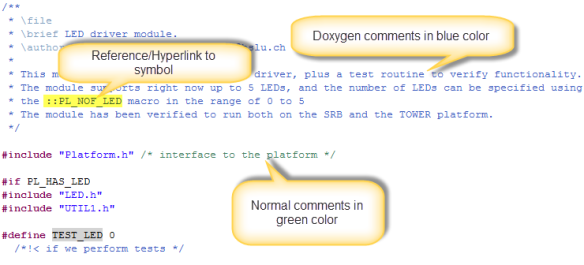
**Syntax Coloring**

Eclox adds a Doxygen workspace default editor to the system:

[](https://mcuoneclipse.files.wordpress.com/2012/06/doxygen-workspace-default.png)

Doxygen Workspace default

With this Doxygen comments are in blue color:

[](https://mcuoneclipse.files.wordpress.com/2012/06/doxygen-source-coloring.png)

Doxygen source coloring

**Doxygen Auto Comments**

What is really cool: Eclox automatically adds comments in Doxgen format. For example I have this in my source:

|  |  |
| --- | --- |
| 1 | int CalcPos(int currPos, int steps); |

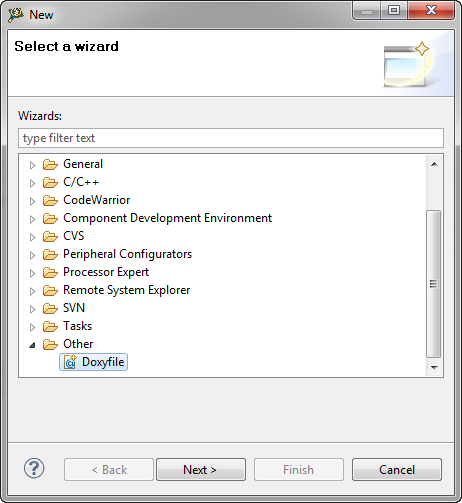
Then I simply can start a Doxygen comment (e.g. with */\*\**) and press ENTER:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | /\*\*  \*  \* @param currPos  \* @param steps  \* @return  \*/    int CalcPos(int currPos, int steps); |

It automatically creates the comment block for me with parameters and return type, so I only need to fill in the content :-).

**New Doxygen Configuration File Wizard**

Doxygen needs a textual configuration file. Eclox adds a wizard to create one using the menu *File > New > Other > Doxyfile*:

[](https://mcuoneclipse.files.wordpress.com/2012/06/file-new-doxyfile.png)

Creating New DoxyFile

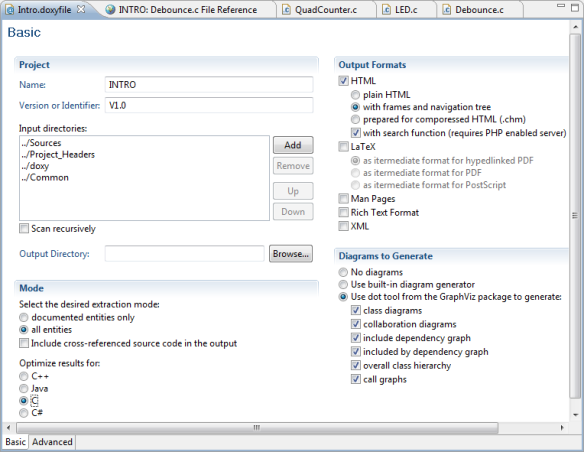
**Doxyfile**

The Doxygen configuration file (*Doxyfile*) is a simple text file which has comments in it to explain the settings:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | # Doxyfile 1.7.2    # 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 = INTRO    # 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.    ... |

**Doxyfile Editor**

Doxygen needs a text configuration file (the *Doxyfile*). With Eclox there is an editor view for that file:

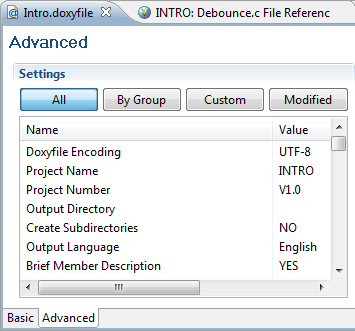
[](https://mcuoneclipse.files.wordpress.com/2012/06/doxyfile-editor.png)

Eclox doxyfile editor

The ‘Basic’ tab of that editor view gives access to the most important settings: The *Input directories* (I list here all the source directories I want to process), the *Output Formats* (I use mostly HTML. For PDF I usually select RTF or LaTeX), *Diagrams* and *Mode* selection.

There is a bug in the Eclox plugin: it looses the language settings for the ‘Optimize results for:’ settings. But that’s not a big deal.

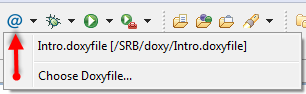
The ‘Advanced’ Tab gives access to all settings:

[](https://mcuoneclipse.files.wordpress.com/2012/06/doxyfile-advanced-tab.png)

Doxyfile Advanced Tab

**Compiling the documentation**

Eclox adds a toolbar button:

[](https://mcuoneclipse.files.wordpress.com/2012/06/eclox-toolbar-button.png)

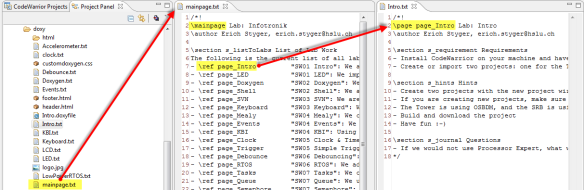
Eclox toolbar button

The button automatically selects the Doxyfile of the current project. Otherwise it is possible to select a different Doxyfile.

Generating the documentation is like normally compiling source files: the Doxygen compiler reports what is going on to the *Console View*, and warnings/errors are reported in the *Problems View*.

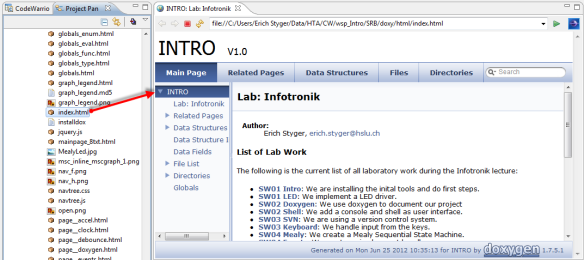
**Other Documentation**

It is possible to document things outside the source files. There are many different commands, *\mainpage* the most important one: it defines the main entry point of the documentation. Using *\page* and *\section* is is possible to organize the documentation, and *\ref* is used to reference labels:

[](https://mcuoneclipse.files.wordpress.com/2012/06/mainpage.png)

Mainpage File with links to other files/pages

If using HTML, then the main entry point for the documentation is the index.html. Eclipse comes with a built in web browser which can be used to open that file. Or use any other HTML viewer:

[](https://mcuoneclipse.files.wordpress.com/2012/06/index-html.png)

index.html as documentation entry point