
**OMII-UK Best Practice:
Software Release Recommendations
Revision: 0.17**

Authors:

Steve Crouch, Rob Baxter, Tim Parkinson

OMII-UK
32/4053
University of Southampton
Southampton
Hants., SO17 1BJ
Tel: +44 (0)23 80598862

Contents

<i>1</i>	<i>Introduction</i>	<i>3</i>
1.1	Purpose of Document.....	3
1.2	Definitions and Relations.....	3
<i>2</i>	<i>Source Release and Build Process</i>	<i>4</i>
2.1	Introduction.....	4
2.2	Objectives.....	4
2.3	The Source Release.....	5
2.3.1	Directory Structure	5
2.3.2	Versioning	5
2.3.3	Contents	6
2.4	Building the Software.....	6
2.4.1	Build Pre-requisites	6
<i>3</i>	<i>Binary Release and Install Process</i>	<i>7</i>
3.1	Introduction.....	7
3.2	Objectives.....	8
3.3	A Binary Release.....	8
3.3.1	Directory Structure	8
3.3.2	Contents	9
3.4	Installing the Software.....	10
3.4.1	Install Pre-requisites	10
<i>4</i>	<i>Configuration and Deployment</i>	<i>10</i>
<i>5</i>	<i>Documentation</i>	<i>11</i>
5.1	Introduction.....	11
5.2	Format.....	11
5.3	High-level Considerations.....	11
5.4	Content.....	11
5.5	Additional Comments.....	12
5.6	Style Guide.....	13
5.6.1	Text	13
5.6.2	Conventions	13
5.6.3	Graphics	13
<i>6</i>	<i>Software Accessibility and Support for Open Development</i>	<i>13</i>
<i>7</i>	<i>References</i>	<i>14</i>

1 Introduction

1.1 Purpose of Document

The target audience for this document is developers wishing to contribute software to OMII-UK. It provides a set of guidelines for:

- Structuring the software for submission, in terms of source code and related software dependencies
- What should be provided in terms of documentation
- What should be expected from software in terms of build, install, configure and ease of use.

The submission of software to OMII-UK that conforms to these guidelines is a prelude to a potential evaluation of that software by OMII-UK that follows the process detailed in the *OMII-UK Software Evaluation Process* [1].

Section 2 describes the OMII-UK guidelines for software releases and the associated build process, Section 3 covers binary releases and related installation processes and Section 4 touches briefly on configuration processes. Section 1.2 below provides definitions of these various terms.

Section 5 provides some guidelines on documentation, and Section 6 describes OMII-UK's policy on support for open development.

1.2 Definitions and Relations

We define the following software states following a software submission to OMII-UK:

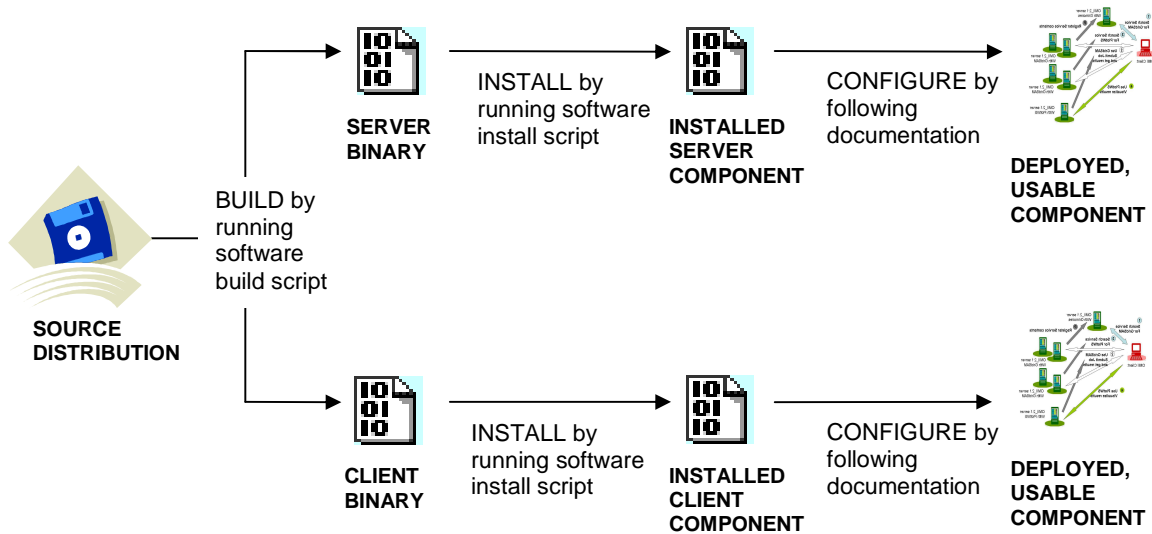
1. **Source Release:** this is a directory structure that contains:
 - a. All the source code and documentation related to that particular software release
 - b. The necessary infrastructure to build the software on a platform.
2. **[Release] Binaries:** these are installable 'components' of the software generated from the source distribution. Typically, the components are either 'client-side' or 'server-side' components:
 - a. They may constitute self-contained installable components for installation elsewhere (perhaps as separate binary releases), **or** be built within the source release for installation on the platform on which the software was built.
 - b. They contain the necessary infrastructure for installing the software on a platform.
3. **Installed Component:** the software has been installed and is ready for configuration within its environment.
4. **Deployed Component:** the software is deployed, configured and is ready to be used.

Where both source and binaries are supplied within a packaged release, the intent of the packaged source and binaries is important. In such a case, building from source may be the primary and recommended route, with binaries provided for simple convenience. Alternatively, the recommended route may be to use the binaries directly, with the source provided mainly for completeness. Product maturity, as well as the nature of the software itself, will influence the preferred methods for use, but in either case this should be made clear in the documentation.

The following three actions facilitate the transition to the next of the above states:

1. **Build (conducted by developer/administrator):** this process generates the component binaries from a source release by running an included build process.
2. **Install (conducted by the end-user/administrator):** an action undertaken to install one of the component binaries respectively by running an included install process.
3. **Configure (conducted by the end-user/administrator):** an action undertaken to configure the software appropriately within its environment according to supplied documentation.

The following illustrates a typical instance of the above:



2 Source Release and Build Process

2.1 Introduction

This section introduces a set of guidelines for structuring and packaging a software source release, and for describing a suitable build process. The objective of the build process is to generate installable binaries from the source material provided in the software source release.

Ideally, these should be self-contained binaries that can be relocated to other platforms for installation, thus not requiring further building of the software, but alternatively the build process can generate code that can be installable from the source release itself on to the same machine. The approach taken should suit the nature of the software, e.g. for an Application Programmers Interface (API)-type product, the generated API libraries may most likely be included within another product, so may not require a separate installable component in its own right. For a product that has significant build, install and configuration requirements, the generation of separate installable binaries may be preferable to separate the build and install processes for administrators and end-users respectively.

2.2 Objectives

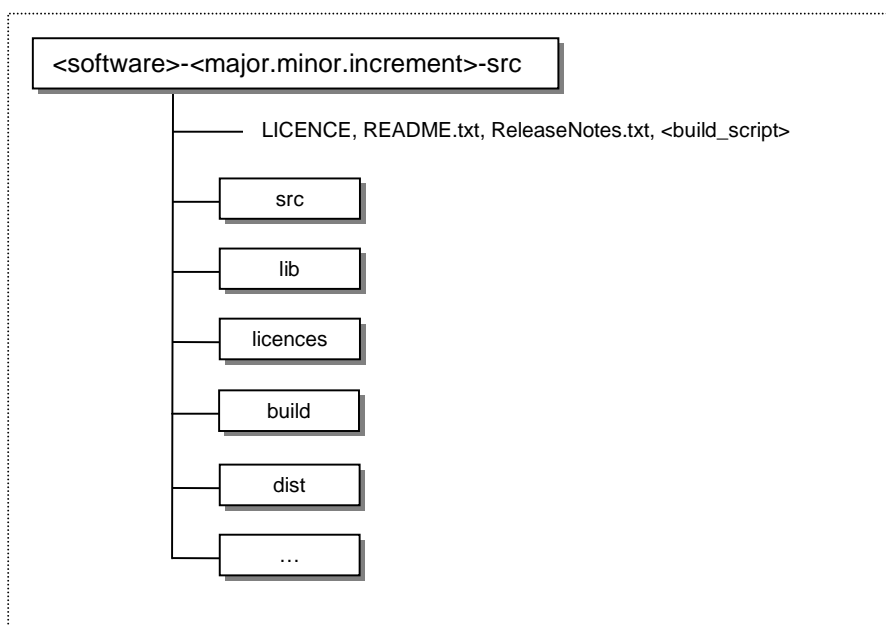
The following overall objectives, in priority order, should be met for a source release by an administrator/user:

- **Easily understood:** documentation should clearly, correctly and concisely explain the build process in terms of pre-requisites, process, and outcomes.
- **Easily buildable:** the release should contain a straightforward, robust build process.
- **Easily maintainable:** the release should be easily modifiable, in case: fixes, enhancements or other changes need to be made to any part of the software in the future.

2.3 The Source Release

2.3.1 Directory Structure

The file/directory structure for the source release should be well organised i.e. separate top-level directories for source, third party libraries, licences, etc., and a minimum of files in the root directory. For example, a suggested directory structure for the source release is as follows:



Suggestions for contents are covered in the next sections. Whilst suggestive, it is recommended that any adopted directory structure appropriately addresses the operation of the software.

2.3.2 Versioning

It is recommended that the source software package is labelled as:

`<software>-<major.minor.increment>-src`

Where software is a string representing the project name with no hyphens, and *major.minor.increment* is the version of the software with the following stipulations:

- No two releases of the software can use the same release version.
- Major stable releases are denoted by an 'even' minor release number, e.g. 1.0.0, 1.2.0.
- Development releases are denoted by an 'odd' minor release number, e.g. 1.1.0, 1.3.0.

- Minor increments may be omitted when equal to zero, e.g. 1.2, 1.3.
- Updates to versions will be released as required (e.g. bug fixes, security patches, etc.) and will be denoted by an increment in the point release i.e. 1.0.0 to 1.0.1. These releases will at a minimum have sufficient testing by developers to verify that the fixes are successful and have not affected established capability.

If a different versioning convention is used, this should be documented.

2.3.3 Contents

LICENCE.txt: the main product licence in UTF-8 text file format covering the software in this release.

README.txt: a UTF-8 text format file that contains general information about the software, e.g. software name, version, authors, packaging date and structure, and contact details. Ideally, it should include anything that a reader requires to build the software, e.g. pre-requisites and links to appropriate documentation.

ReleaseNotes.txt: a version history that contains a list of changes in this and prior releases (minimally since the last major release), and the platform and environment used for testing.

<build_script>: the included build script (e.g. Maven pom.xml, Ant build.xml, Make files, Bash/Perl script) that is used to build the software.

The suggested static directories are as follows:

- *src/*: contains all source code, e.g. Java, C, JSP, HTML, LaTeX source, etc.
- *lib/*: required third-party libraries
- *licences/*: licences for third-party libraries

NB: A **strict requirement** is that the third-party licences and LICENCE.txt file are included.

Optionally, the following directories are dynamically created during the build process:

- *build/*: intermediate build files are stored here during the build process
- *dist/*: final, installable files are stored here following successfully running the build process, either packaged into a form ready for installation elsewhere (i.e. archived into .zip or .tar.gz), or in a form ready for installation on the machine used for building the software.

2.4 Building the Software

Assuming the platform, environmental and software pre-requisites have been met, this should be a simple automated process. Ideally, this would take the form of running a single command or set of commands, e.g. running *mvn* or *ant* in the unpacked source directory for Maven or Ant respectively.

In addition, the process should be idempotent, in that performing subsequent builds produces the same output.

2.4.1 Build Pre-requisites

In general, these should be kept to a minimum. In addition to any others required, the following should be covered:

Pre-requisite	Ideal	Acceptable	Version Info. From
Platform	Either RedHat Enterprise Linux (RHEL) / RedHat Academic Server (AS) version 4 or above and/or Windows XP SP3, Windows Vista	One or more of RedHat 4+, Fedora 5+, SuSE 10+, Windows XP SP2, Windows Vista	Linux: <i>uname -a</i> and contents of <i>/etc/issue</i>
Environment	Clear and concise instructions for configuring the build environment e.g. environment variables, system configuration changes		Linux: build-specific environment variables from <i>env</i>
Source compiler	Java: Sun Java SE Development Kit (JDK) versions 5 & 6 Perl: version 5.8 or above Python: version 2.3.0 or above C/C++: Gnu C Compiler (gcc) version 3.x or above	Java: Sun Java SE Development Kit (JDK) versions 5 Perl: version 5.8 or above Python: version 2.3.0 or above C/C++: Gnu C Compiler (gcc) version 3.x or above	Java: <i>java -version</i> C/C++: <i>gcc -v</i> Perl: <i>perl -version</i> Python: <i>python -v</i>
Build infrastructure	Maven build script (version 2.0.2 or above) Ant build script (version 1.7.1 or above) C/C++ Make (version 3.80 or above)	Command-line scripts in Bash or Perl (version 5 or above)	Maven: <i>mvn -version</i> Ant: <i>ant -version</i> Perl: <i>perl -version</i> Make: <i>make -version</i>
Required third-party components	Small – e.g. libraries: included within source release where technically and legally permissible, in appropriately redistributable source or binary form – MUST comply with any licenses and where necessary include copies of these licenses.	Clear and concise instructions for downloading and integrating particular versions of components into the build	
	Large – e.g. other CSP components: clear and concise instructions for downloading and integrating particular versions of components into the build		

Within the documentation (see section 5.4), source build pre-requisites should be clearly stated in terms of the software/platforms/infrastructure versions and their required configuration.

3 Binary Release and Install Process

3.1 Introduction

This section introduces a set of guidelines for structuring and packaging a software binary release, and for describing a suitable install process. The objective of the install process is to utilise installable binaries generated from the build process and install them into their intended operating environment for use.

3.2 Objectives

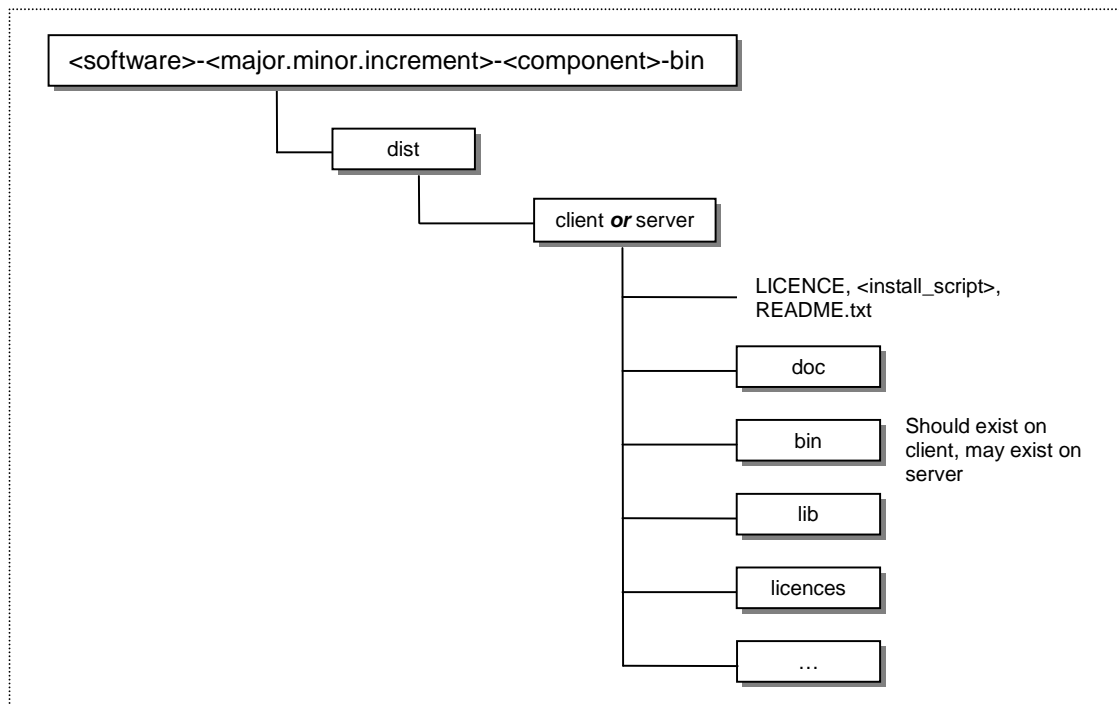
The following overall objectives, in priority order, should be met for a binary release by an administrator/user:

- **Easily understood:** documentation should clearly and concisely explain the software, its intended purpose and how to use and configure it. It should also explain the install process in terms of pre-requisites, process, and outcomes. See section 5 for documentation guidelines and suggestions.
- **Easily installable:** the release should contain a straightforward, automated, robust install process.
- **Easily configurable:** the installation should be easily configurable to adapt its function to suit its environment and intended use.
- **Easily uninstallable:** the release should contain either a straightforward, robust uninstall process. Ideally this should be automated, although a well-documented manual process is sufficient.
- **Migratability/upgradability:** where not feasible to provide an automated process, a well documented process should exist for:
 - Migrating an existing deployment (i.e. all relevant configuration and data persistence therein) to another deployment location
 - Upgrading an existing deployment from a previous version of the software.

3.3 A Binary Release

3.3.1 Directory Structure

The file/directory structure for the package should be well organised i.e. separate top-level directories for documentation, binaries, third party libraries, licences, etc., and a minimum of files in the root directory. For example, a suggested directory structure for binary releases is as follows:



Therefore, upon building the software from source, the `dist/` directory is created that contains separate directories for each built part of the software, e.g. `client` and/or `server` (ready for install on either the build machine or packaged ready for installation elsewhere). Whilst suggestive, it is recommended that any adopted directory structure appropriately addresses the operation of the software.

3.3.2 Contents

LICENCE.txt: the main product licence in ASCII text file format covering the software in this release.

README.txt: an ASCII text format file that contains general information about the software e.g. software name, version, authors, packaging date and structure, and contact details. Ideally, it should include anything that a reader requires to install the software, e.g. pre-requisites and links to appropriate documentation.

ReleaseNotes.txt: a version history that contains a list of changes in this and prior releases (minimally since the last major release),, and the platform and environment used for testing.

<install_script>: the included install script (e.g. Ant `build.xml`, Bash/Perl script) that is used to install the software.

The suggested static directories are as follows:

- `doc/`: contains all ready-to-use documentation, e.g. HTML, JavaDoc, etc.
- `bin/`: contains built executable programs
- `lib/`: required third-party libraries
- `licences/`: licences for third-party libraries

NB: A **strict requirement** is that the third-party licences and `LICENCE.txt` file are included.

3.4 Installing the Software

Assuming the platform, environmental and software pre-requisites have been met, this should be a simple automated process. Ideally, this would take the form of running a single command or set of commands, e.g. running *ant* or an install shell script in the unpacked binary directory.

Displayed output should concisely capture the progress, and any installation errors should be clear and concise, ideally with recommended remedial action.

3.4.1 Install Pre-requisites

In general, these should be kept to a minimum. In addition to any others required, the following should be covered:

Pre-requisite	Ideal	Acceptable	Version Info. From
Platform	Either RedHat Enterprise Linux (RHEL) / RedHat Academic Server (AS) version 4 or above and/or Windows XP SP3, Windows Vista	One or more of RedHat 4+, Fedora 5+, SuSE 10+, Windows XP SP2, Windows Vista	Linux: <i>uname -a</i> and contents of <i>/etc/issue</i>
Environment	Clear and concise instructions for configuring the install environment, e.g. environment variables, system configuration changes		Linux: install-specific environment variables from <i>env</i>
Runtime environment	Java: Sun Java Runtime Environment (JRE) versions 5 & 6 C/C++: libc version 3.x or above	Java: Sun Java Runtime Environment (JRE) versions 5 C/C++: libc version 3.x or above	Java: <i>java -version</i> C/C++: <i>gcc -v</i>
Install infrastructure	Ant build script (version 1.6.1 or above)	Command-line scripts in Bash or Perl (version 5 or above)	Ant: <i>ant -version</i> Perl: <i>perl -version</i>
Required third-party components	Small – e.g. libraries: included within binary release where technically and legally permissive, in binary form	Clear and concise instructions for downloading and integrating particular versions of components into the release	
	Large – e.g. other CSP components: clear and concise instructions for downloading and integrating particular versions of components into the install		

Within the documentation (see section 5.4), binary install pre-requisites should be clearly stated in terms of the software/platforms/infrastructure versions and their required configuration.

4 Configuration and Deployment

It may be necessary to configure the software after installation prior to use. These steps should be clearly documented, with examples in each case.

In OMII terms, this forms the final Configuration process for the installed software, and results in a Deployed Component or set of Components.

5 Documentation

5.1 Introduction

It is difficult to provide guidelines for writing the documentation that accompanies a software component, because the structure and content of the documentation is ultimately dependent on the functionality of the software and the requirements of the intended user. However, this section contains some basic suggestions for the structure and content that can be followed.

5.2 Format

Preferably, documentation should be provided in both HTML and PDF formats. Rendered HTML should be viewable as intended on popular versions of major web browsers favoured at the time of release. For PDF, the page size should be A4.

5.3 High-level Considerations

In general, the following questions should be addressed before writing the documentation:

1. What problem does the software solve?
2. How does the software solve the problem?
3. How does the user/administrator install and configure (and optionally build) the software? How can a user quickly access the functionality of the software?
4. How can a developer make use, or extend the functionality, of the software?
5. What is the current release state, and how does the current release differ from earlier releases?

The documentation should provide the answers to all of the above questions.

5.4 Content

The content of the documentation will vary, dependent on the type of software and its purpose. However, certain subjects are universal, and the following should be considered.

Software should include the following minimum documentation:

- Summary:
 - Project name, language, software domain (service, client, tools)
 - What the software does and why the user would want to use it
 - Project and technical contact information
- Building the Software
 - Skill pre-requisites – the technical or other skills required to build the software
 - Software pre-requisites – see section 2.3 (including details for obtaining and configuring any required third-party components)
 - Build guide
 - Details for obtaining and configuring an required third-party components
- Installing and Configuring the Software
 - Skill pre-requisites – the technical or other skills required to install and configure the software
 - Pre-requisites – see section 3.2 (including details for obtaining and configuring any required third-party components)
 - Detailed installation and configuration guide
 - Administration guide for advanced configuration

- Using the Software
 - Skill pre-requisites – the technical or other skills required to use the software
 - Basic user guide – how to use the software to perform basic tasks for general use cases
 - Advanced user/developer guide – how to use the software's more advanced features (e.g. develop using a provided API, how to configure and use for domain-specific use cases, etc).

- Product Roadmap: the envisioned development of the product for the foreseeable future, in terms of releases, functionality and support. A link may be provided to other documentation or a web resource that provides the roadmap.

Including the following is highly encouraged:

- Getting Started: a short 'read me' or getting started/quickstart guide with link to details of software pre-requisites, required elements of the OMII software, user guides, etc. that describes how to get a simple working example up and running.

Ideally, documentation should also include (in order of desirability):

- Support
 - Tutorials and examples
 - Sample applications & development guides
 - API documentation
 - 1-3 promotional slides in A4 landscape PDF format, ideally with the original PowerPoint .ppt or OpenOffice .odp slides that provide a brief overview of the product
 - These slides should be suitable for disseminating to others.

- Product Overview and Architecture:
 - Requirements specification, e.g. system overview, user cases, functional specification, relationships to external components, security requirements, design and implementation constraints, specifications, standards
 - Software design description, e.g. design decisions and rationale, architecture design, decomposition and interrelationship between components, how the design satisfies the requirements, components input and output, overview of data flow and control flow
 - Interface design description, e.g. types of interface (data transfer or control), format of individual data elements provided by interfaces, protocols used by the interface, interface compatibility

- Testing and Quality Assurance
 - Documentation of testing strategy
 - Description of test environment (operating system etc.)
 - Instructions detailing how to run the tests
 - Test results and analysis
 - Overall assessment of the software, as demonstrated by the test results
 - Any remaining deficiencies, limitations or constraints detected by testing

5.5 Additional Comments

References to external sources, hyperlinks, etc. should be included, where possible, to allow the user to obtain further information if required.

Cross-referencing to relevant sections of the documentation should be used wherever possible. It is particularly important to implement cross-referencing when listing long sets of instructions, or when discussing complex inter-related concepts that are described in different sections of the documentation.

5.6 Style Guide

The following is a suggested basic style guide.

5.6.1 Text

The text used in the documentation should comprise:

1. A maximum of three header sizes, all in the same font (e.g. Arial, left aligned, one line space after each paragraph)
2. One font for body text, which should be the same as the header font (e.g. Arial, Justified, one line space after each paragraph)
3. One constant-width font for code (e.g. Courier, left aligned, no line space after each paragraph)

5.6.2 Conventions

Where possible, the conventions used in OMII-UK's documentation should be repeated in the documentation, i.e.

[UNIX] denotes Unix specific commands
[Windows] denotes Windows specific commands
Bold used for file and directory names

5.6.3 Graphics

Where possible, graphics should be supplied in png, jpg or gif formats.

5.6.4 Video

Where possible, any supplied video should be in platform independent Flash .swf or mpg formats, as opposed to Windows Media Video .wmv which is only easily viewable on Windows.

6 Software Accessibility and Support for Open Development

Support for an open development process is a major requirement for software that is supported by OMII-UK. The software has to be open source and adhere to a suitable open source licence. To support this open development the following should be provided:

- **Source code repository:** developers should be able to access either an up-to-date CVS or Subversion (SVN) code repository for source code. A policy for committing updates to the code base should also be available for those wishing to contribute to improving the software. See [2] and [3] for instructions and guidelines for working with CVS and SVN respectively.
- **Packaged, usable software:** the binary releases should be packaged for immediate use in a suitable archive format, e.g. .tar.gz for Linux, .zip for Windows.
- **Overview web page:** download access for the software should also be provided on a web page that includes an introduction to the software and how to get started.
- **Public bug/issue tracker:** users/developers should be able to view existing software issues and add their own.

- **Mailing lists:** software announce and developer mailing lists for those wishing to be kept informed as to releases and developments of the software, or wishing to join discussions.

All of these should have a lifetime beyond the funded part of the project, be publically accessible and frequently updated. A good example of how this could be provided is through a project in SourceForge (<http://www.sourceforge.net>), which is able to provide all of these facilities and is highly configurable.

7 References

- [1] *OMII-UK Software Evaluation Process*, S. Crouch *et al*, OMII-UK, August 2009.
- [2] *Open Source Development with CVS, 3rd Edition*, Karl Fogel and Mosche Bar, 2003.
- [3] *Version Control with Subversion, 2nd Edition*, Ben Collins-Sussman, Brian W. Fitzpatrick & C. Michael Pilato, 2008.