INL/RPT-24-77165
Revision 0

ORCA Software Quality Assurance Plan

March | 2024

Ross D. Hays
Takanori Kajihara
Linyu Lin
Paul W. Talbot

Integrated Energy Systems

INL/RPT-24-77165
Revision 0

ORCA Software Quality Assurance Plan

Ross D. Hays
Takanori Kajihara
Linyu Lin
Paul W. Talbot

March | 2024March | 2024

Idaho National Laboratory
Integrated Energy Systems
Idaho Falls, Idaho 83415

http://www.ies.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

ABSTRACT

This document summarizes the software quality assurance (SQA) planning activities conducted for the Optimization of Real-time Capacity Allocation (ORCA) plug-in. It outlines the approaches and document structures adopted to maintain high standards of software quality. It also gives examples of certain SQA documents.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

Page intentionally left blank

ACRONYMS

EA	Enterprise Architecture
EDMS	Electronic Document Management System
INL	Idaho National Laboratory
LST	Configuration Item List
NQA	Nuclear Quality Assurance
ORCA	Optimization of Real-Time Capacity Allocation
QA	quality assurance
QLD	Quality Level Determination
RAVEN	Risk Analysis and Virtual Environment
SRS	Software Requirements Specifications
SSD	Safety Software Determination
SQA	software quality assurance

Page intentionally left blank

ORCA Software Quality Assurance Plan

1. INTRODUCTION

The Optimization of Real-Time Capacity Allocation (ORCA) plug-in serves as an advanced modeling toolkit for enhancing the efficiency of real-time control and mathematical optimization for digital twins. It encompasses a broad range of virtual facility models, actual physical facilities, and their interconnections, thus facilitating the use of these virtual models to optimally control the physical facilities. The core objective of this software is to bridge the gap between virtual planning and real-world execution, ensuring that decisions made in the virtual environment can be seamlessly translated into tangible actions performed in physical facilities.

This document is dedicated to detailing and elucidating the comprehensive efforts undertaken to implement software quality assurance (SQA) practices for the ORCA software. It outlines the strategic approaches adopted to maintain high standards of software quality, including rigorous testing methodologies, adherence to best practices in software development, continuous integration and deployment processes, and robust documentation. The aim is to ensure that the ORCA plug-in remains reliable, efficient, and effective in its role of optimizing physical facility operations by using digital twins. Through meticulous SQA efforts, this document demonstrates Idaho National Laboratory (INL)'s commitment to delivering a toolset that meets all expectations for quality, performance, and reliability in real-time control and optimization applications.

1.1. Purpose and Scope

This document details the processes and documentation set out to formalize the SQA pedigree for the ORCA plugin to the Risk Analysis and Virtual Environment (RAVEN) software platform. To the greatest extent possible, these processes and documentation leverage the existing RAVEN plug-in quality assurance (QA) processes, which were developed to comply with the provisions established in ASME "Quality Assurance Requirements for Nuclear Facility Applications" (NQA-1) (2009 edition) [1], Subpart 2.7, "Quality Assurance Requirements for Computer Software for Nuclear Facility Applications." And though applying these processes to the development of software is an acceptable way to ensure that the software offers adequate assurance of safety in its designated roles in nuclear analyses, no such application is currently under consideration in the present scope. Instead, these QA standards were selected as representing the gold standard in quality but also permitting a graded approach to documentation, and because they leverage considerable institutional expertise and existing procedural infrastructure.

It must be noted that the purpose of this current scope of work is not to develop a fully NQA-1-compliant software product, but to instead set out the necessary developmental framework and procedures to enable subsequent development of such a product. This scope sets out both the systematic approach that is to be employed in the development of ORCA, as well as the documentary artifacts required to demonstrate adherence to said approach. The described approach leverages the existing SQA plan for RAVEN, along with industry standard tools and INL-specific document review and archiving systems. The present report is laid out as follows. Section 2 defines some key terms, Section 3 explains the ORCA development process, and Section 4 describes the required QA artifacts, their purposes, and any required attributes or processing steps they entail.

2. DEFINITIONS

- **Baseline.** A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for use and further development, and that can be changed only by using an approved change control process. [ASME NQA-1-2018, with the NQA-1a-2009 addenda edited[1]]

- **Validation.** Confirmation, through the provision of objective evidence (e.g., acceptance test), that the requirements for a specific intended use or application have been fulfilled. [ISO/IEC/IEEE 24765:2017(E) edited [2]]
- **Verification.** The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [ISO/IEC/IEEE 24765:2017(E) edited [2]]

3. ORCA DEVELOPMENT PROCESS

The NQA-1 standards require that a systematic approach be utilized for software engineering activities. The suggested software development lifecycle encompasses the following four steps (reflected in Fig. 1), as taken from the RAVEN SQA plan (PLN-5552) [5]:

1. Software requirements
2. Software design
3. Software design implementation (coding)
4. Testing.

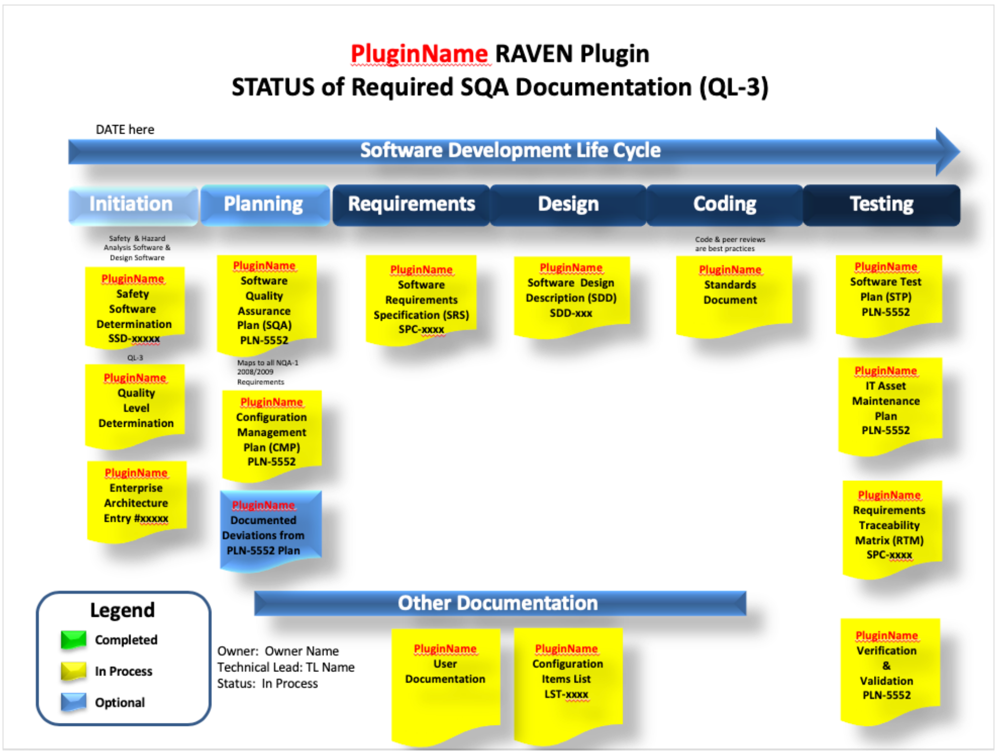

PluginName RAVEN Plugin
STATUS of Required SQA Documentation (QL-3)
DATE here
Software Development Life Cycle
Initiation
Planning
Requirements
Design
Coding
Testing
Safety & Hazard Analysis Software & Design Software
PluginName Safety Software Determination SSD-xxxxx
QL-3
PluginName Quality Level Determination
PluginName Enterprise Architecture Entry #xxxxx
PluginName Software Quality Assurance Plan (SQA) PLN-5552
Maps to all NQA-1 2008/2009 Requirements
PluginName Configuration Management Plan (CMP) PLN-5552
PluginName Documented Deviations from PLN-5552 Plan
PluginName Software Requirements Specification (SRS) SPC-xxxx
PluginName Software Design Description (SDD) SDD-xxx
Code & peer reviews are best practices
PluginName Standards Document
PluginName Software Test Plan (STP) PLN-5552
PluginName IT Asset Maintenance Plan PLN-5552
PluginName Requirements Traceability Matrix (RTM) SPC-xxxx
PluginName Verification & Validation PLN-5552
Other Documentation
PluginName User Documentation
PluginName Configuration Items List LST-xxxx
Legend
Completed
In Process
Optional
Owner: Owner Name
Technical Lead: TL Name
Status: In Process

Figure 1. RAVEN plugin development process and documentation (https://github.com/idaholab/raven/wiki/external-plugin-SQA-requirements).

While the formality and rigor of such a systematic approach is often eschewed in the earliest stages of software development, this type of approach must be applied prior to starting the processes for formalizing the initial "qualified" release. The RAVEN SQA plan (PLN-5552) sets out the framework for the development process and the schedule for submitting the required documents. Certain of these documents (e.g., PLN-5552) are to be utilized as-is, while others are to be specially developed. Of the developed documents, those pertaining to the initialization stage will be created in their respective business systems and archived in the INL Electronic Document Management System (EDMS) (itself an NQA-1-compliant document archive). The documents for the subsequent software engineering steps will be developed concurrently with the ORCA software in the ORCA GitHub repository, using the Sphinx documentation support software. These documents will come in two different formats, each utilized according to its respective use case. The first and most useful format is an interactive webpage that allows users to easily browse the interlinked pages to find the information they need. The second and more durable format is a PDF/A rendering of the same content, suitable for formal review and archiving purposes aimed at meeting the document retention requirements of NQA-1. These PDFs will be archived in EDMS and will be reviewed and archived with each new major release. Further details on the layout and formatting of these documents are provided in Section 4.

The initiation steps will establish the anticipated use of the ORCA software, including the required level of quality rigor necessary to support its nexus to safety. Having established the necessary rigor, subsequent development steps may be implemented to meet the given standard. These development steps will include enumeration of the requirements that the developed software shall meet, the formal design and implementation of the software, and the subsequent testing to demonstrate that the software meets all aforementioned requirements. Each of these steps will output a document that must be reviewed and archived for each new major release of the software. The required reviews are to be performed by an individual who is competent in the technology but was not involved in creating the document. The review itself will be performed and documented using the INL Document Change Request system, which is itself an NQA-1-compliant mechanism for such reviews.

4. SQA DOCUMENT STRUCTURE

Any software product that utilizes the PLUGIN system within RAVEN and is compliant with the RAVEN SQA process can reference the RAVEN SQA plan, procedures, and documentation. As a plugin to the RAVEN software platform, the ORCA SQA references PLN-5552, “RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan,” which defines the SQA processes to be implemented and covers the following SQA documents:

- SQA plan
- Configuration management plan
- Standards document
- Software test plan
- Information technology asset management plan
- Verification and validation plan.

In addition to the above-listed SQA documents, the documents listed in Table 1 are produced to support the QA pedigree of the ORCA software.

Table 1. ORCA QA documents.

Document	Purpose	Development Source	Status
Safety Software Determination (SSD)	Formally determines the software’s impact on nuclear safety	INL Workflow	
Quality Level Determination (QLD)	Formally determines the level of quality rigor required to meet the safety requirements determined in the SSD.	INL Workflow	
Enterprise Architecture (EA) Entry	Registers the software for subsequent management and tracking.	INL ServiceNow	

Document	Purpose	Development Source	Status
Software Requirements Specifications (SRS)	Documents the requirements of the software.	Sphinx	In development
Requirement Traceability Matrix	Shows the linkage between the requirements above and the test cases or documents that verify the requirements have been satisfactorily met.	Sphinx	In development
Configuration Item List (LST)	A comprehensive inventory of the ORCA software components, required libraries, supporting tools, and documentation.	Sphinx	In development
Software Design Description	Provides documentation as to the key components of the ORCA software, how they interact, and what their limitations are.	Sphinx	In development
Deviations	Documents any aspects of the ORCA development process that differ from those prescribed in PLN-5552, and offers evidence of their adequacy to meet the relevant QA standards.	Sphinx	In development
User Documentation	Provides users with ample information on how to utilize ORCA, interpret the results, and gauge the scope of validity of those results.	Sphinx	In Progress
Developer Documentation	Informs developers of the expectations for developing the software so that it can be included in the ORCA main branch.	Sphinx	In Progress

SRS, Requirement Traceability Matrix, Deviations, and Software Design Description are expected to be fully developed for the current version in June 2024. LST and User Documentation for the current version are expected to be fully developed in September 2024.

The SSD evaluates how well the software application align with the U.S. Department of Energy's safety software definition. The QLD assesses software application quality by asking questions, and EA Entry maintains an inventory of every software application, detailing their various attributes (e.g., name, version, technical contacts, associated SSD and QLD, and corresponding software management plans). SSD, QLD, and EA Entry are stored in either EDMS or the EA entry. SRS outlines the requirements of software slated to be developed for application. This includes functional requirements, usability requirements, and system interfaces. The Requirement Traceability Matrix provides a way to keep track of all the requirements defined in SRS and ensure that they are continually met at each stage of the development lifecycle. The Software Design Description outlines the architecture and detailed design of the software system to be developed. It serves as a comprehensive blueprint describing how the software will be structured and how it will behave, affording proof that the specified requirements outlined in SRS will indeed be met. LST includes a comprehensive list of files and items that are under configuration control for ORCA (e.g., source code, test case files, documentation managed via Sphinx, stationary documentation, and third-party library identification). The Deviation document clarifies all deviations from PLN-5582. All these documents are and will continue to be stored in the ORCA [6] GitHub repository (https://github.com/idaholab/ORCA).

The ORCA SQA document was created in the Sphinx format [7]. Sphinx is a powerful documentation generator that converts restructured text files into HTML websites and other formats. It is widely used to create comprehensive, well-organized documentation for software projects. Sphinx's versatility stems from its extensive support for modular extensions, enabling features such as automatic generation of content from source code, indexing, and cross referencing. This capability makes it an ideal tool for maintaining detailed documentation and keeping it up to date with code changes. The files are stored in the ORCA repository and can be viewed on the Sphinx-built webpage by utilizing the Read the Docs software [8]. Examples of the ORCA documentation webpage and the type of PDF file it generates are attached as appendices to this document.

5. REFERENCES

[1] ASME. 2009. "Quality Assurance Requirements for Nuclear Facility Applications." American Society of Mechanical Engineers, NQA 1 2008 with the NQA-1a-2009 addenda, First Edition.

[2] ISO/IEC/IEEE. 2017. "Systems and software engineering Vocabulary." ISO/IEC/IEEE 24765:2017€, Second Edition.

[3] LWP 13620, "Managing Information Technology Assets."

[4] SDD-513, "RAVEN Software Design Description." https://github.com/idaholab/raven/tree/devel/doc/sqa

[5] PLN-5552, "RAVEN and RAVEN Plug-ins Software Quality Assurance and Maintenance and Operations Plan." https://github.com/idaholab/raven/tree/devel/doc/sqa

[6] Talbot, P., et al. 2023. "Optimization Of Real-time Capacity Allocation." Idaho National Laboratory. https://doi.org/10.11578/dc.20230815.1.

[7] Turner A., Tran B., Sewell C., Freitag F., Andersen J. L., Burnon J., Finucane S., Shimizukawa T., Komiya T., "Sphinx," GitHub, [Online]. Available: https://github.com/sphinx-doc/sphinx. [Accessed 15 March 2024].

[8] "Read the Docs," GitHub, [Online]. Available: https://github.com/readthedocs. [Accessed 15 March 2024].

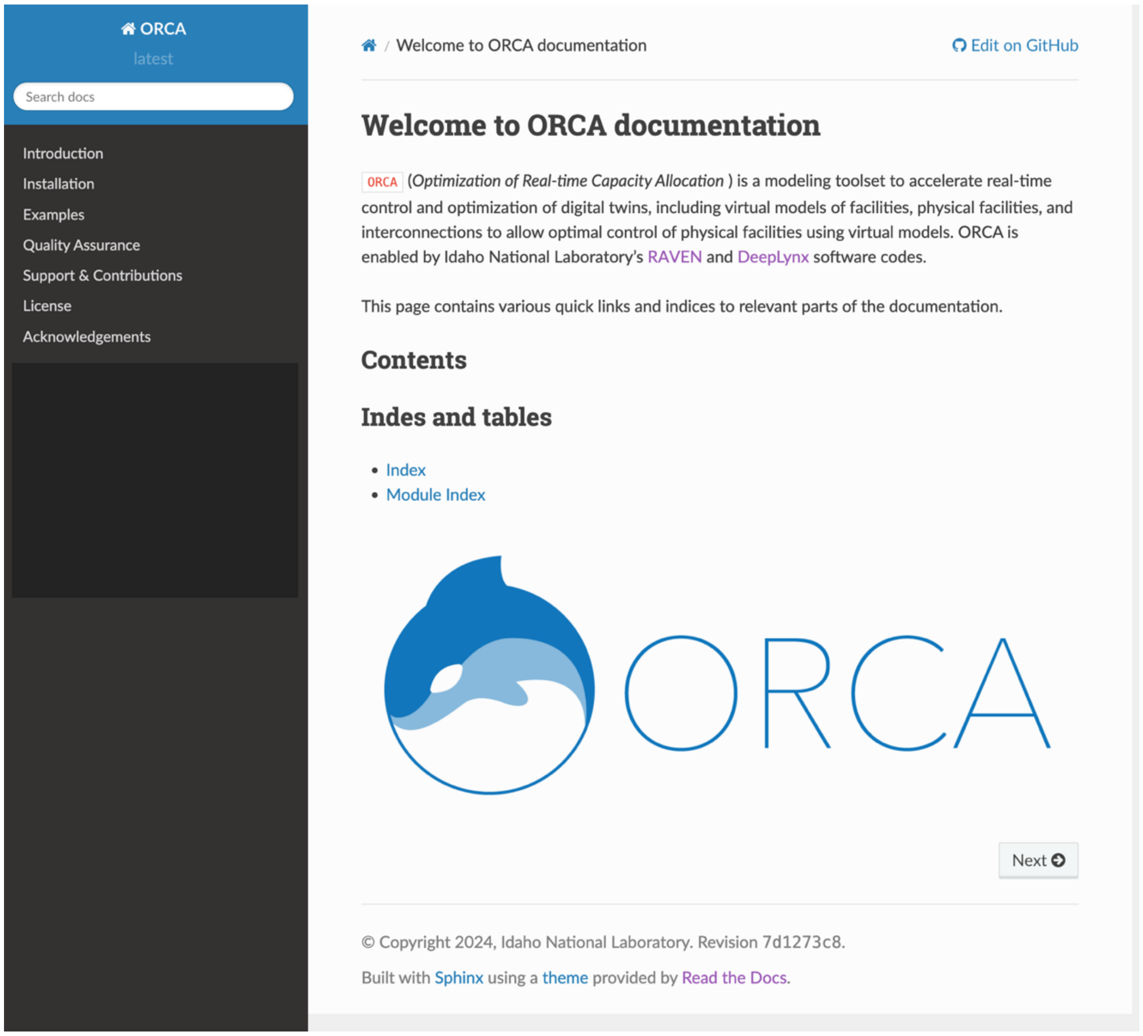
Page intentionally left blank

Appendix A
Examples of Documentation Using Sphinx

Page intentionally left blank

Appendix A
Examples of Documentation Using Sphinx

A.1 ORCA documentation page built with Sphinx



Welcome to ORCA documentation

ORCA (*Optimization of Real-time Capacity Allocation*) is a modeling toolset to accelerate real-time control and optimization of digital twins, including virtual models of facilities, physical facilities, and interconnections to allow optimal control of physical facilities using virtual models. ORCA is enabled by Idaho National Laboratory's RAVEN and DeepLynx software codes.

This page contains various quick links and indices to relevant parts of the documentation.

Contents

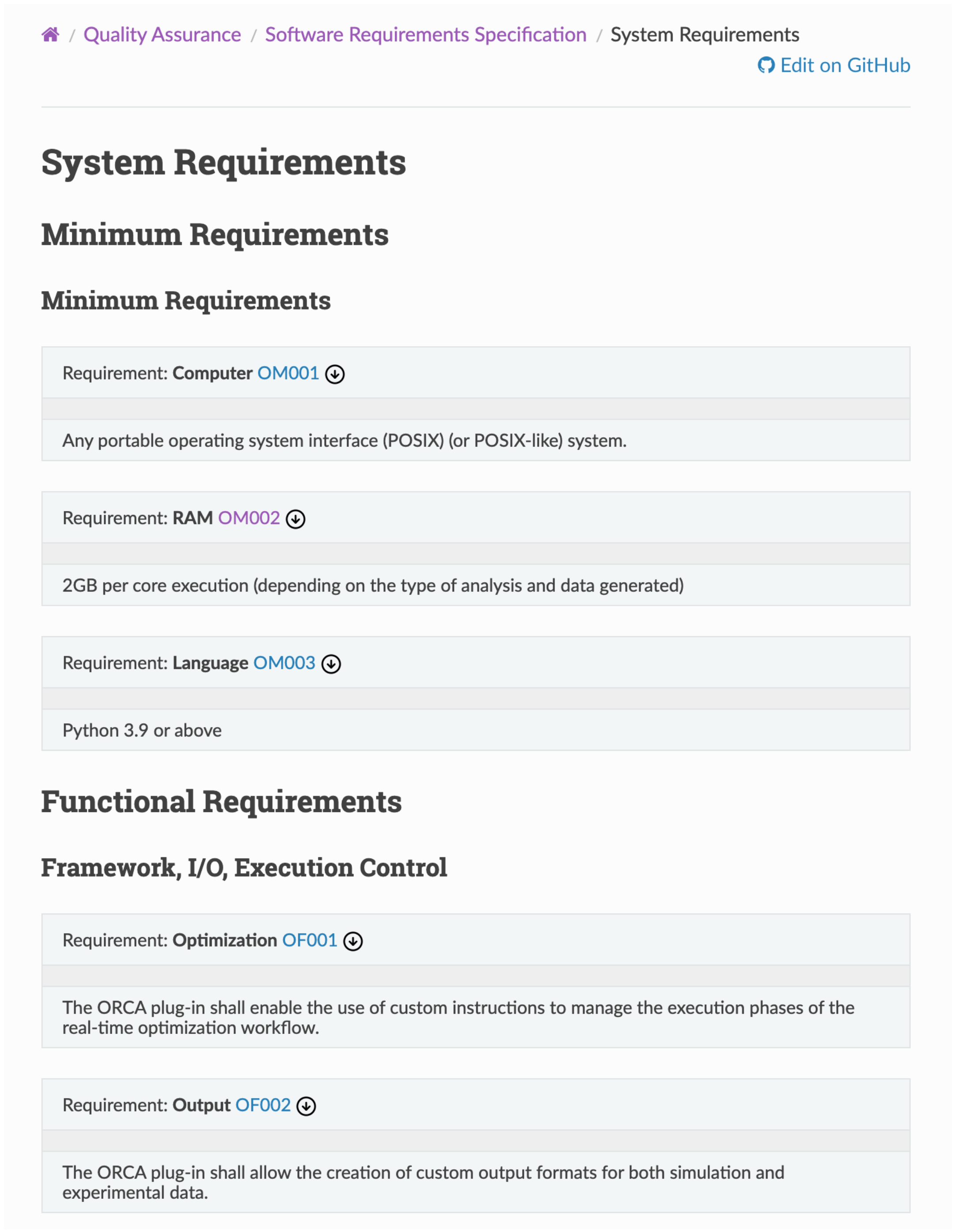
Indes and tables

- Index
- Module Index

© Copyright 2024, Idaho National Laboratory. Revision 7d1273c8.

Built with Sphinx using a theme provided by Read the Docs.

Figure A-1. ORCA documentation using Sphinx.



/ Quality Assurance / Software Requirements Specification / System Requirements

Edit on GitHub

System Requirements

Minimum Requirements

Minimum Requirements

Requirement: **Computer** OM001

Any portable operating system interface (POSIX) (or POSIX-like) system.

Requirement: **RAM** OM002

2GB per core execution (depending on the type of analysis and data generated)

Requirement: **Language** OM003

Python 3.9 or above

Functional Requirements

Framework, I/O, Execution Control

Requirement: **Optimization** OF001

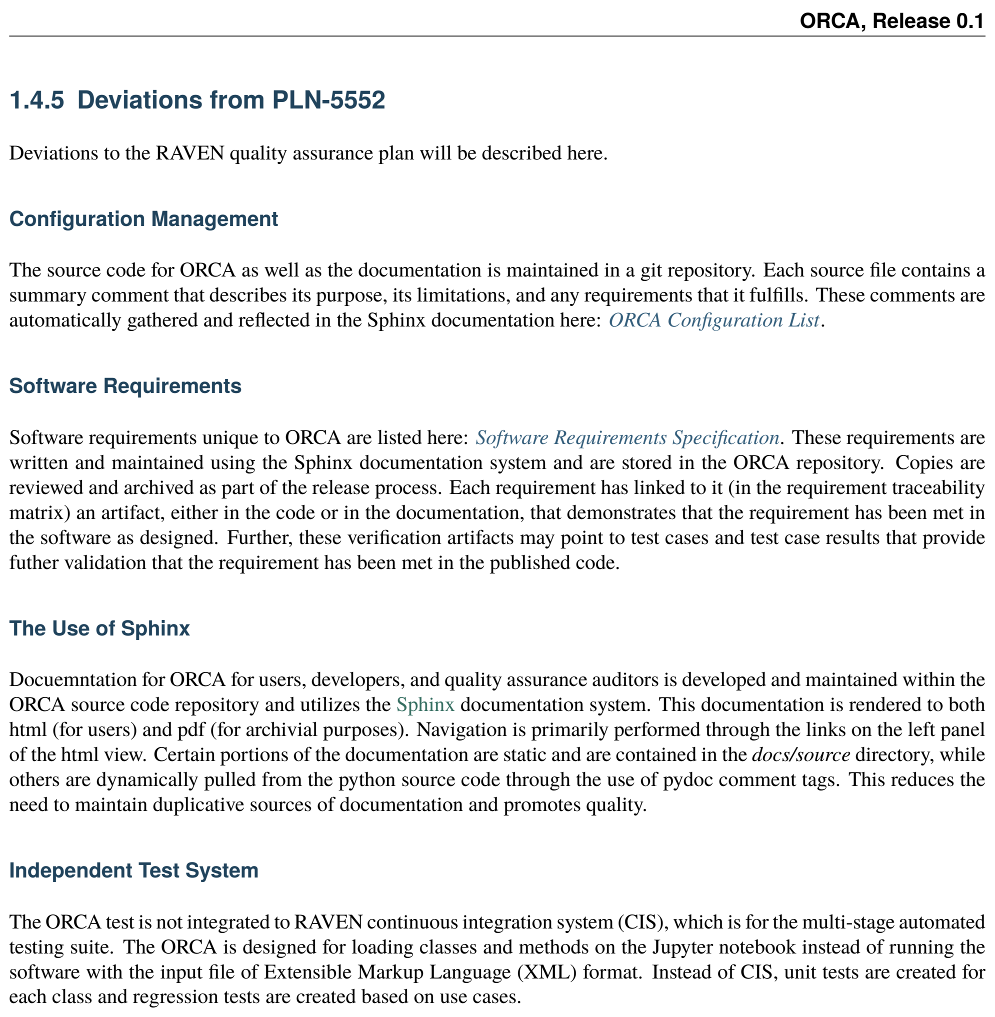
The ORCA plug-in shall enable the use of custom instructions to manage the execution phases of the real-time optimization workflow.

Requirement: **Output** OF002

The ORCA plug-in shall allow the creation of custom output formats for both simulation and experimental data.

Figure A-2. SRS page in the ORCA documentation.

A.2 PDF files generated from the Sphinx-based webpage



1.4.5 Deviations from PLN-5552

Deviations to the RAVEN quality assurance plan will be described here.

Configuration Management

The source code for ORCA as well as the documentation is maintained in a git repository. Each source file contains a summary comment that describes its purpose, its limitations, and any requirements that it fulfills. These comments are automatically gathered and reflected in the Sphinx documentation here: *ORCA Configuration List*.

Software Requirements

Software requirements unique to ORCA are listed here: *Software Requirements Specification*. These requirements are written and maintained using the Sphinx documentation system and are stored in the ORCA repository. Copies are reviewed and archived as part of the release process. Each requirement has linked to it (in the requirement traceability matrix) an artifact, either in the code or in the documentation, that demonstrates that the requirement has been met in the software as designed. Further, these verification artifacts may point to test cases and test case results that provide futher validation that the requirement has been met in the published code.

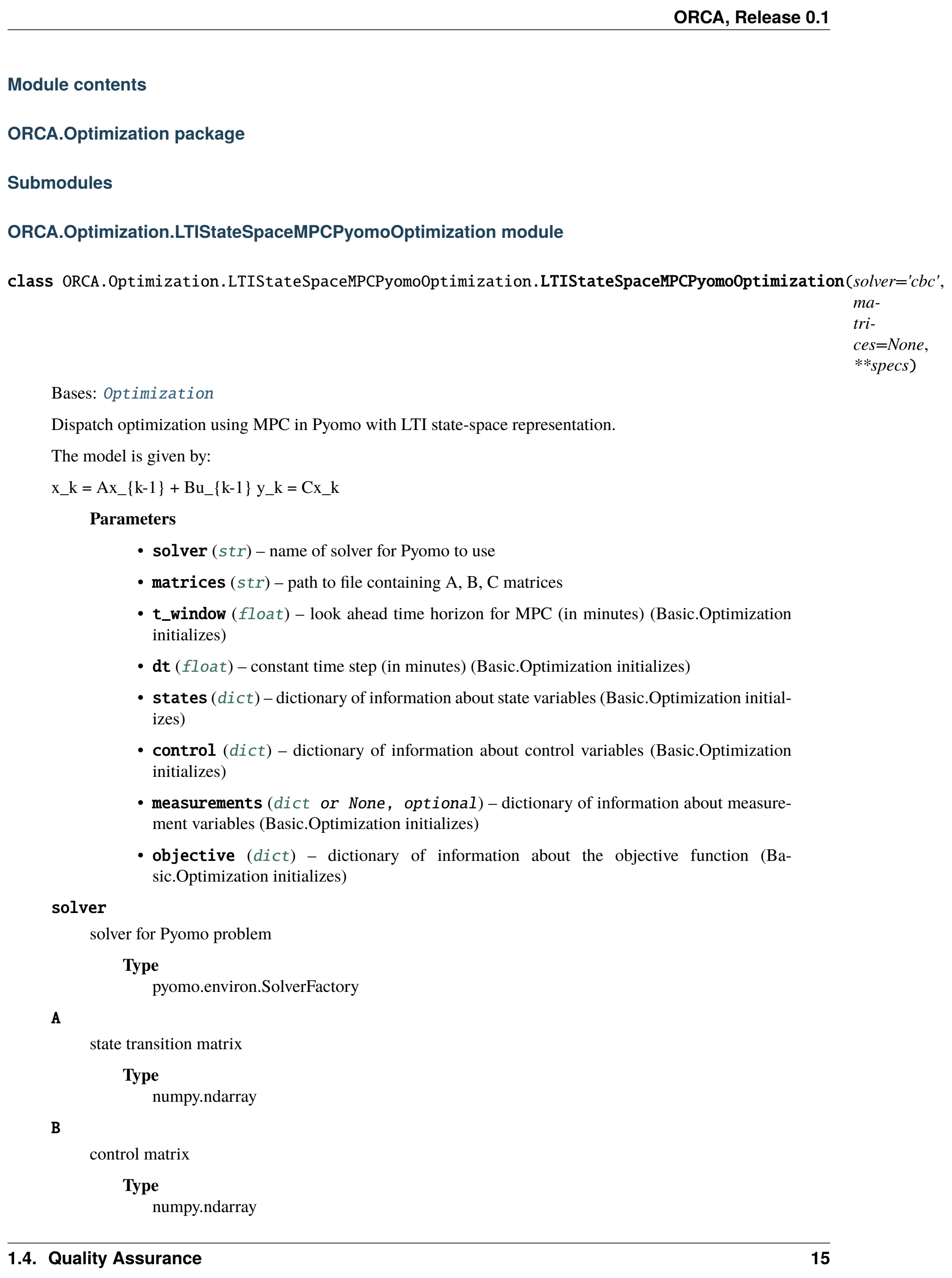
The Use of Sphinx

Docuemntation for ORCA for users, developers, and quality assurance auditors is developed and maintained within the ORCA source code repository and utilizes the Sphinx documentation system. This documentation is rendered to both html (for users) and pdf (for archivial purposes). Navigation is primarily performed through the links on the left panel of the html view. Certain portions of the documentation are static and are contained in the *docs/source* directory, while others are dynamically pulled from the python source code through the use of pydoc comment tags. This reduces the need to maintain duplicative sources of documentation and promotes quality.

Independent Test System

The ORCA test is not integrated to RAVEN continuous integration system (CIS), which is for the multi-stage automated testing suite. The ORCA is designed for loading classes and methods on the Jupyter notebook instead of running the software with the input file of Extensible Markup Language (XML) format. Instead of CIS, unit tests are created for each class and regression tests are created based on use cases.

Figure A-3. SRS page in the ORCA documentation.



ORCA, Release 0.1

Module contents

ORCA.Optimization package

Submodules

ORCA.Optimization.LTIStateSpaceMPCPyomoOptimization module

class `ORCA.Optimization.LTIStateSpaceMPCPyomoOptimization.`**`LTIStateSpaceMPCPyomoOptimization`**(*solver='cbc'*, *matrices=None*, ***specs*)

Bases: `Optimization`

Dispatch optimization using MPC in Pyomo with LTI state-space representation.

The model is given by:

x_k = Ax_{k-1} + Bu_{k-1} y_k = Cx_k

Parameters

- **`solver`** (`str`) – name of solver for Pyomo to use
- **`matrices`** (`str`) – path to file containing A, B, C matrices
- **`t_window`** (`float`) – look ahead time horizon for MPC (in minutes) (Basic.Optimization initializes)
- **`dt`** (`float`) – constant time step (in minutes) (Basic.Optimization initializes)
- **`states`** (`dict`) – dictionary of information about state variables (Basic.Optimization initializes)
- **`control`** (`dict`) – dictionary of information about control variables (Basic.Optimization initializes)
- **`measurements`** (`dict or None, optional`) – dictionary of information about measurement variables (Basic.Optimization initializes)
- **`objective`** (`dict`) – dictionary of information about the objective function (Basic.Optimization initializes)

`solver`

solver for Pyomo problem

Type
pyomo.environ.SolverFactory

`A`

state transition matrix

Type
numpy.ndarray

`B`

control matrix

Type
numpy.ndarray

1.4. Quality Assurance 15

Figure A-4. ORCA module list in LST.