StrictDoc

Software for writing technical requirements and specifications

Project goals:	Technical documentation and requirements management
Documents storage:	Plain text files
Export formats:	HTML, RST/Sphinx, ReqIF, PDF, Excel
License model:	Open source software, Apache 2 license
Project page:	https://github.com/strictdoc-project/strictdoc
Release date:	April, 2024
Version:	0.0.55

Contents

1 User Guide . . . 1
1.1 Introduction . . . 1
1.1.1 Contact the developers . . . 1
1.2 Examples . . . 2
1.2.1 Hello World . . . 2
1.2.2 StrictDoc Examples repository . . . 3
1.2.3 StrictDoc Templates repository . . . 3
1.2.4 Other examples . . . 3
1.3 Installing StrictDoc . . . 3
1.3.1 Requirements . . . 3
1.3.2 Installing StrictDoc as a Pip package (recommended way) . . . 3
1.3.3 Installing “nightly” StrictDoc as a Pip package . . . 4
1.3.4 Installing StrictDoc into a Docker container . . . 4
1.3.5 Installing StrictDoc as a Snap package (not maintained) . . . 4
1.4 Running StrictDoc . . . 4
1.4.1 Static HTML export . . . 4
1.4.2 Web server . . . 4
1.4.3 Security considerations . . . 5
1.5 IDE support . . . 5
1.6 SDoc syntax . . . 6
1.6.1 Document structure . . . 6
1.6.1.1 Strict rule #1: One empty line between all nodes . . . 7
1.6.1.2 Strict rule #2: No content is allowed outside of SDoc grammar . . . 7
1.6.1.3 Strict rule #3: No empty strings . . . 7
1.6.2 Grammar elements . . . 8
1.6.2.1 Document . . . 8
1.6.2.1.1 Document configuration options . . . 8
1.6.2.1.1.1 ENABLE_MID . . . 9
1.6.2.1.1.2 MARKUP . . . 9
1.6.2.1.1.3 AUTO_LEVELS . . . 9
1.6.2.1.1.4 REQUIREMENT_STYLE . . . 9
1.6.2.1.1.5 REQUIREMENT_IN_TOC . . . 10
1.6.2.2 Requirement . . . 10
1.6.2.2.1 UID . . . 11
1.6.2.2.2 Level . . . 11
1.6.2.2.3 Status . . . 11
1.6.2.2.4 Tags . . . 11
1.6.2.2.5 Relations (previously REFS) . . . 11
1.6.2.2.5.1 Requirement relation roles . . . 12
1.6.2.2.6 Title . . . 13
1.6.2.2.7 Statement . . . 13
1.6.2.2.8 Rationale . . . 13
1.6.2.2.9 Comment . . . 13

1.6.2.3 Section . . . 14
1.6.2.3.1 Nesting sections . . . 14
1.6.2.3.2 Free text . . . 15
1.6.2.3.3 Section without a level . . . 15
1.6.2.4 Composing documents from other documents . . . 16
1.6.2.5 Composite requirement . . . 17
1.6.3 Machine identifiers (MID) . . . 18
1.6.4 Custom grammars . . . 18
1.6.4.1 Supported field types . . . 19
1.6.4.2 Reserved fields . . . 21
1.6.4.3 Relations . . . 21
1.6.4.3.1 Relation roles . . . 22
1.6.4.3.2 Parent vs Child relations . . . 22
1.6.4.4 Importing grammar from grammar file . . . 23
1.6.5 Links . . . 24
1.6.5.1 Section links . . . 24
1.6.5.2 Anchors . . . 24
1.6.5.2.1 Anchor example . . . 24
1.7 Search and filtering . . . 24
1.7.1 Query engine . . . 25
1.7.2 Filtering content . . . 25
1.8 Markup . . . 25
1.8.1 Images . . . 26
1.8.2 Mathjax support . . . 26
1.9 Export formats . . . 26
1.9.1 HTML documentation tree by StrictDoc . . . 26
1.9.1.1 Standalone HTML pages . . . 27
1.9.2 HTML export via Sphinx . . . 27
1.9.3 PDF export via Sphinx/LaTeX . . . 27
1.9.4 JSON . . . 28
1.10 Manage project tree . . . 28
1.10.1 Automatic assignment of requirements UID . . . 28
1.11 Traceability between requirements and source code . . . 28
1.12 ReqIF support . . . 29
1.12.1 Import flow (ReqIF -> SDoc) . . . 30
1.12.2 Export flow (SDoc -> ReqIF) . . . 30
1.12.3 ReqIF options . . . 30
1.12.4 ReqIF implementation details . . . 30
1.13 Excel support . . . 31
1.13.1 Import flow (Excel XLS -> SDoc) . . . 31
1.13.2 Export flow (SDoc -> Excel XLSX) . . . 31
1.14 Options . . . 31
1.14.1 Project-level options . . . 31
1.14.1.1 Project title . . . 32
1.14.1.2 Path to assets . . . 32
1.14.1.3 Path to source root . . . 32
1.14.1.4 Include/exclude document paths . . . 32

1.14.1.5 Include/exclude source files paths . . . 33
1.14.1.6 Selecting features . . . 33
1.14.1.6.1 Enable all features . . . 34
1.14.1.6.2 Disable all features . . . 34
1.14.1.7 Server configuration . . . 34
1.14.1.7.1 Host and port . . . 34
1.14.2 Command-line interface options . . . 35
1.14.2.1 Project title . . . 35
1.14.2.2 Parallelization . . . 35
1.15 Python API . . . 35
1.16 Portability considerations . . . 36
1.17 Experimental features . . . 36
1.17.1 Project statistics screen . . . 36
1.17.2 HTML2PDF document generator . . . 37
1.17.3 Mermaid diagramming and charting tool . . . 38
1.17.4 Shadow features . . . 38
1.18 StrictDoc's limitations . . . 38
1.18.1 Limitations of RST support by StrictDoc . . . 38
1.18.2 Limitations of web user interface . . . 39
1.18.2.1 Concurrent use of web user interface . . . 39
1.19 Known issues . . . 39
1.19.1 Exporting document free text to ReqIF and vice versa . . . 40
1.19.2 Running out of semaphores on macOS . . . 40
2 F.A.Q. . . . 41
2.1 What is StrictDoc? . . . 41
2.2 Resources about StrictDoc . . . 41
2.3 Which web server is recommended for StrictDoc documentation? . . . 41
2.4 Is StrictDoc compatible with Sphinx? . . . 42
2.5 How did the SDoc text language become what it is? . . . 42
2.6 How StrictDoc compares to other tools? . . . 44
2.6.1 Doorstop . . . 44
2.6.2 Sphinx . . . 44
2.6.3 Sphinx-Needs . . . 45
2.6.4 FRET . . . 45
2.7 How long has the StrictDoc project been around? . . . 46
2.8 Which StrictDoc statistics are available? . . . 46
3 Development Plan . . . 47
3.1 Project goals . . . 47
3.2 Project milestones . . . 47
3.2.1 The roadmap diagram . . . 48
3.3 Verification . . . 49
3.4 Python baseline . . . 49
4 Release Notes . . . 50
4.1 0.0.55 (2024-04-28) . . . 50
4.2 0.0.54 (2024-04-17) . . . 50
4.3 0.0.53 (2024-04-01) . . . 50
4.4 0.0.52 (2024-03-25) . . . 51

4.5 0.0.51 (2024-03-20) . . . 51
4.6 0.0.50 (2024-03-19) . . . 51
4.7 0.0.49 (2024-03-11) . . . 51
4.8 0.0.48 (2024-01-24) . . . 52
4.9 0.0.47 (2023-11-20) . . . 53
5 Contributing to StrictDoc . . . 54
5.1 Contributor checklist . . . 54
5.2 How can I help? . . . 54
5.2.1 Spread the word . . . 54
5.2.2 ReqIF users . . . 54
5.2.3 TeX / LaTeX / Sphinx experts . . . 55
6 Developer Guide . . . 56
6.1 Getting started . . . 56
6.1.1 System dependencies . . . 56
6.1.1.1 Windows-specific: Long Path support . . . 56
6.1.2 Installing StrictDoc from GitHub (developer mode) . . . 56
6.2 Invoke for development tasks . . . 57
6.3 Main “Check” task . . . 57
6.4 Python code . . . 57
6.5 Git workflow . . . 58
6.6 Frontend development . . . 58
6.7 Running End-to-End Web tests . . . 59
6.8 Running integration tests . . . 59
6.9 Documentation . . . 59
6.10 Conventions . . . 59
7 Requirements Tool Specification (L1) . . . 60
7.1 Summary of user needs . . . 60
7.1.1 Free and open source tool . . . 60
7.1.2 Document types . . . 60
7.1.2.1 Document structure differences . . . 61
7.1.3 Workflows . . . 61
7.1.4 Target audience . . . 62
7.2 Documentation management . . . 63
7.2.1 Documents (CRUD) . . . 63
7.2.2 Browsing documentation tree . . . 64
7.2.3 Documents with nested sections/chapters structure . . . 64
7.2.4 Assembling documents from fragments . . . 64
7.2.5 Document meta information (UID, version, authors, signatures, etc) . . . 65
7.2.6 Document versioning . . . 65
7.2.7 Text formatting capabilities . . . 65
7.3 Requirements management . . . 66
7.3.1 Requirements CRUD . . . 66
7.3.2 Minimal requirement field set . . . 66
7.3.3 Custom fields . . . 67
7.3.4 Structuring requirements in documents . . . 67
7.3.5 Move requirement nodes within document . . . 68
7.3.6 Move nodes between documents . . . 68

7.3.7 Auto-provision of Requirement UIDs . . . 68
7.3.8 Link requirements together . . . 69
7.3.9 Multiple link roles . . . 69
7.3.10 Reverse parent links . . . 69
7.3.11 Unique identification of requirements . . . 70
7.3.12 Requirements database consistency checks . . . 70
7.3.13 Requirement syntax validation (e.g. EARS) . . . 70
7.4 Tool configurability . . . 71
7.4.1 Project-level configuration . . . 71
7.4.2 Document-level configuration . . . 71
7.5 Performance . . . 72
7.5.1 Support large requirements sets . . . 72
7.5.2 Support large project trees . . . 72
7.6 Data integrity . . . 73
7.6.1 Data integrity of documentation/requirements . . . 73
7.7 Existing workflows . . . 73
7.7.1 Excel-like viewing and editing of requirements . . . 73
7.7.2 1000-feet view . . . 74
7.7.3 Traceability matrices . . . 74
7.7.4 Compliance matrices . . . 74
7.7.5 Requirements coverage . . . 75
7.7.6 Progress report . . . 75
7.7.7 Change management . . . 76
7.8 Usability, installation and usage . . . 76
7.8.1 General usability . . . 76
7.8.2 Easy user experience . . . 77
7.8.3 Support projects with a large number of users . . . 77
7.8.4 Individual use (home PC) . . . 77
7.8.5 Server-based deployments (IT-friendly setup) . . . 78
7.8.6 Requirements database . . . 78
7.8.7 Programming access via API (Web) . . . 78
7.8.8 Programming access via API (SDK) . . . 79
7.8.9 Programmatic access to requirements data . . . 79
7.9 Implementation suggestions . . . 79
7.9.1 Static HTML export . . . 79
7.9.2 Graphical user interface (GUI) . . . 80
7.9.3 Command-line interface . . . 80
7.9.4 Web API interface . . . 80
7.9.5 Version control (Git) . . . 80
7.9.6 Support major operating systems . . . 81
7.9.7 Conservative languages for implementation . . . 81
7.9.8 Long-term maintainability of a tool . . . 82
7.10 Text-based requirements language (optional) . . . 82
7.10.1 Text files for storing documentation and requirements . . . 82
7.10.2 Strict text language syntax . . . 83
7.10.3 Machine-readable format . . . 83
7.10.4 Requirements data from multiple repositories . . . 83

7.11 Requirements and source code . . . 83
7.11.1 Traceability between requirements and source code . . . 83
7.12 Requirements exchange formats (export/import) . . . 84
7.12.1 ReqIF export/import . . . 84
7.12.2 CSV export/import . . . 85
7.12.3 Excel export/import . . . 85
7.13 Collaboration on requirements . . . 85
7.13.1 Support user accounts . . . 85
7.13.2 Send notifications about updated requirements . . . 85
7.14 Development process . . . 86
7.14.1 Requirements engineering . . . 86
7.14.2 Self-hosted requirements . . . 86
7.14.3 Test coverage . . . 86
7.14.4 Tool qualification . . . 87
7.15 Licensing and distribution . . . 88
7.15.1 Open source . . . 88
7.15.2 Only open source dependencies . . . 88
7.15.3 Free . . . 88
8 StrictDoc Requirements Specification (L2) . . . 89
8.1 SDoc data model . . . 89
8.1.1 Data model . . . 89
8.1.2 Requirement model . . . 89
8.1.3 Requirement model fields . . . 90
8.1.4 Requirement model default fields . . . 90
8.1.5 Document model . . . 90
8.1.6 Document metadata . . . 91
8.1.7 Section model . . . 91
8.1.8 Free text . . . 91
8.1.9 Composeable document . . . 92
8.1.10 Requirement relations . . . 92
8.1.11 Requirement relation roles . . . 92
8.2 SDoc text markup . . . 93
8.2.1 SDoc markup language . . . 93
8.2.2 Identical SDoc content by import/export roundtrip . . . 93
8.2.3 SDoc and Git storage . . . 93
8.2.4 SDoc file extension . . . 94
8.2.5 One document per one SDoc file . . . 94
8.2.6 Fixed grammar . . . 94
8.2.7 Default grammar fields . . . 95
8.2.8 Custom grammar / fields . . . 95
8.2.9 Importable grammars . . . 95
8.2.10 UID identifier format . . . 96
8.2.11 Support RST markup . . . 96
8.2.12 MathJAX . . . 96
8.2.13 No indentation . . . 96
8.2.14 Type-safe fields . . . 97

8.3 Graph database . . . 97
8.3.1 Traceability index . . . 97
8.3.2 Uniqueness UID in tree . . . 97
8.3.3 Detect links cycles . . . 98
8.3.4 Link document nodes . . . 98
8.3.5 Automatic resolution of reverse relations . . . 98
8.4 Documentation tree . . . 99
8.4.1 Finding documents recursively . . . 99
8.5 Web/HTML frontend . . . 99
8.5.1 General export requirements . . . 99
8.5.1.1 Export to static HTML website . . . 99
8.5.1.2 Web interface . . . 100
8.5.1.3 Export to printable HTML pages (HTML2PDF) . . . 100
8.5.1.4 Preserve generated file names . . . 100
8.5.2 Screen: Project tree . . . 100
8.5.2.1 View project tree . . . 100
8.5.2.2 Create document . . . 101
8.5.2.3 Delete document . . . 101
8.5.3 Screen: Document (DOC) . . . 101
8.5.3.1 Read document . . . 101
8.5.3.2 Update document . . . 101
8.5.3.3 Edit requirement nodes . . . 102
8.5.3.4 Move requirement / section nodes within document . . . 102
8.5.3.5 Edit Document grammar . . . 102
8.5.3.6 Edit Document options . . . 102
8.5.3.7 Auto-generate requirements UIDs . . . 103
8.5.3.8 Buttons to copy text to buffer . . . 103
8.5.4 Screen: Table (TBL) . . . 103
8.5.4.1 View TBL screen . . . 103
8.5.5 Screen: Traceability (TR) . . . 103
8.5.5.1 View TR screen . . . 103
8.5.6 Screen: Deep traceability (DTR) . . . 104
8.5.6.1 View DTR screen . . . 104
8.5.7 Screen: Project statistics . . . 104
8.5.7.1 Display project statistics . . . 104
8.5.8 Screen: Traceability matrix . . . 104
8.5.8.1 Traceability matrix . . . 104
8.5.9 Screen: Project tree diff . . . 105
8.5.9.1 Project tree diff . . . 105
8.6 Requirements-to-source traceability . . . 105
8.6.1 Link requirements with source files . . . 105
8.6.2 Annotate source file . . . 105
8.6.3 Single-line code marker . . . 106
8.6.4 Generate source coverage . . . 106
8.6.5 Generate source file traceability . . . 106

8.7 Export/import formats . . . 107
8.7.1 RST . . . 107
8.7.1.1 Export to RST . . . 107
8.7.1.2 Docutils . . . 107
8.7.2 ReqIF . . . 107
8.7.2.1 Export/import from/to ReqIF . . . 107
8.7.2.2 Standalone ReqIF layer . . . 108
8.7.3 Excel and CSV . . . 108
8.7.3.1 Export to Excel . . . 108
8.7.3.2 Selected fields export . . . 108
8.7.4 Graphviz/Dot export . . . 109
8.7.4.1 Export to Graphviz/Dot . . . 109
8.8 Command-line interface . . . 109
8.8.1 General CLI requirements . . . 109
8.8.1.1 Command-line interface . . . 109
8.8.2 Command: Manage . . . 109
8.8.2.1 Command: Auto UID . . . 109
8.8.2.1.1 Auto-generate requirements UIDs . . . 109
8.9 Python API . . . 110
8.9.1 StrictDoc Python API . . . 110
8.10 Web server . . . 110
8.10.1 Web server . . . 110
8.11 User experience . . . 110
8.11.1 Strict mode by default . . . 110
8.11.1.1 Warnings are errors . . . 110
8.12 Configurability . . . 111
8.12.1 strictdoc.toml file . . . 111
8.12.2 Feature toggles . . . 111
8.12.3 ‘Host’ parameter . . . 111
8.13 Performance . . . 112
8.13.1 Process-based parallelization . . . 112
8.13.2 Caching of parsed SDoc documents . . . 112
8.13.3 Incremental generation of documents . . . 112
8.13.4 Caching of RST fragments . . . 113
8.13.5 On-demand loading of HTML pages . . . 113
8.13.6 Precompiled Jinja templates . . . 113
8.14 Development process requirements . . . 114
8.14.1 General process . . . 114
8.14.1.1 Priority handling of critical issues in StrictDoc . . . 114
8.14.2 Requirements engineering . . . 114
8.14.2.1 Requirements-based development . . . 114
8.14.2.2 Self-hosted requirements . . . 114
8.14.3 Implementation requirements . . . 115
8.14.3.1 Programming languages . . . 115
8.14.3.1.1 Python language . . . 115
8.14.3.2 Cross-platform availability . . . 115
8.14.3.2.1 Linux . . . 115

8.14.3.2.2 macOS . . . 115
8.14.3.2.3 Windows . . . 116
8.14.4 Implementation constraints . . . 116
8.14.4.1 Use of open source components . . . 116
8.14.4.2 No heavy UI frameworks . . . 116
8.14.4.3 No use of NPM . . . 116
8.14.4.4 No use of JavaScript replacement languages (e.g., Typescript) . . . 117
8.14.4.5 Monolithic application with no microservices . . . 117
8.14.4.6 No reliance on containerization . . . 117
8.14.5 Coding constraints . . . 118
8.14.5.1 Use of asserts . . . 118
8.14.5.2 Use of type annotations in Python code . . . 118
8.14.6 Linting . . . 118
8.14.6.1 Compliance with Python community practices (PEP8 etc) . . . 118
8.14.7 Static analysis . . . 119
8.14.7.1 Static type checking . . . 119
8.14.8 Testing . . . 119
8.14.8.1 Unit testing . . . 119
8.14.8.2 CLI interface black-box integration testing . . . 119
8.14.8.3 Web end-to-end testing . . . 119
8.14.8.4 At least one integration or end-to-end test . . . 120
8.15 Code hosting and distribution . . . 120
8.15.1 Code hosting . . . 120
8.15.1.1 GitHub . . . 120
8.15.2 StrictDoc license . . . 121
9 Design Document . . . 122
9.1 Overview . . . 122
9.2 Building blocks . . . 122
9.3 High-level architecture . . . 122
9.4 StrictDoc command-line application . . . 123
9.5 StrictDoc web application . . . 123
9.5.1 The HTML Over the Wire (Hotwire) architecture . . . 124
9.6 Parsing SDoc files . . . 124
10 StrictDoc Backlog . . . 125
10.1 StrictDoc challenges . . . 125
10.1.1 Real-time editing out of scope . . . 125
10.2 Backlog . . . 125
10.2.1 Auto-commit to Git repository . . . 125
10.2.2 Auto-generate section UIDs . . . 125
10.2.3 Screen: Project home . . . 126
10.2.3.1 View project home page . . . 126
10.2.4 Screen: Traceability navigator . . . 126
10.2.4.1 Traceability navigator . . . 126
10.2.5 Formal modeling . . . 126
10.2.5.1 Integration with other systems engineering processes . . . 126
10.2.5.2 Integration with Capella . . . 126
10.2.5.3 Support STPA method . . . 127

10.2.5.4 Formalized statements . . . 127
10.2.5.5 AI Assistant . . . 127
10.2.6 LaTeX export . . . 127
10.2.6.1 Export to Tex . . . 127
10.2.7 Focused mode: Edit a single section / requirement . . . 127
10.2.8 Interoperability with Doxygen . . . 128
10.2.9 Fuzzy search (the whole documentation) . . . 128
10.2.10 Derived requirements . . . 128
10.2.11 Support Markdown markup . . . 128
10.2.12 Language Server Protocol (LSP) . . . 128
10.2.13 UML . . . 128
10.2.14 Export/import to CSV . . . 129
10.2.15 Web API . . . 129
10.2.16 Multi-user workflow . . . 129
10.2.16.1 Multi-user editing of documents . . . 129
10.2.16.2 User accounts . . . 130
10.2.16.3 Update notifications . . . 130
10.2.17 Requirement validation according to EARS syntax . . . 130
10.2.18 WYSIWYG editing . . . 130
10.2.19 Tables HTML editor . . . 131
10.2.20 Move requirement / section nodes between documents . . . 131
10.2.21 Auto-completion for requirements UIDs . . . 131
10.2.22 Attach image to requirement . . . 131
10.2.23 Provide contextual help about RST markup . . . 132
10.2.24 TBL: Hide/show columns . . . 132
10.2.25 TBL: Select/deselect tags . . . 132
10.2.26 Screen: Impact analysis . . . 132
10.2.26.1 Impact analysis . . . 132
10.2.27 ReqXLS . . . 133
10.3 Backlog: Web-based user interface . . . 133
10.4 Backlog: Nice to have . . . 133
10.5 Backlog: Technical debt . . . 134
10.6 Open questions . . . 134
10.6.1 One or many input sdoc trees . . . 134
11 Credits . . . 136
11.1 Contributions to StrictDoc . . . 136
11.2 Open source software . . . 136
11.3 Hosting and Continuous Integration . . . 137
11.4 Commercial IDEs by JetBrains . . . 137
12 Technical Note: DO-178C requirements tool requirements . . . 138
12.1 Already implemented features . . . 138
12.1.1 Document concept . . . 138
12.1.2 Strict specified grammar . . . 138
12.1.3 Requirement UID autocompletion . . . 139
12.1.4 Multiple git repositories document assembly . . . 139
12.1.5 Document fragments in separate files . . . 139
12.1.6 PDF and HTML publishing . . . 140

12.1.7 Graphical user interface (GUI) . . . 140
12.1.8 Configuration: ‘Host’ parameter . . . 140
12.1.9 No use of proprietary technology . . . 141
12.1.10 Source file coverage . . . 141
12.1.11 Project-level grammar . . . 141
12.2 Needs discussion . . . 142
12.2.1 WYSIWYG editing . . . 142
12.2.2 Diff between document trees . . . 142
12.2.3 Traceability matrices . . . 142
12.2.4 Impact analysis . . . 143
12.2.5 Uncovered requirement report . . . 143
12.2.6 Interoperability with Sphinx . . . 144
12.2.7 Multi-user editing of documents . . . 144
12.2.8 Support for Derived requirements . . . 145
13 Technical Note: Zephyr requirements tool requirements . . . 146
13.1 Multiple files / include mechanism . . . 146
13.2 Clear separation of requirements (machine-readable) . . . 146
13.3 Custom fields . . . 146
13.4 Links . . . 147
13.5 Multiple link roles . . . 147
13.6 ReqIF export . . . 147
13.7 CSV . . . 148
13.8 Unique ID management . . . 148
13.9 Text formatting capabilities . . . 148
13.10 Minimal requirement field set . . . 149
13.11 Requirements to source code traceability . . . 149
13.12 Non-intrusive links in source code . . . 150
13.13 Structuring requirements in documents . . . 150
13.14 Status field . . . 150
13.15 Tool Qualifiability . . . 151

1. User Guide

1.1 Introduction

StrictDoc is software for technical documentation and requirements management.

Summary of StrictDoc features:

- The documentation files are stored as human-readable text files.
- A simple domain-specific language DSL is used for writing the documents. The text format for encoding this language is called SDoc (strict-doc).
- StrictDoc reads `*.sdoc` files and builds an in-memory representation of a document tree.
- From this in-memory representation, StrictDoc can generate the documentation into a number of formats including HTML, RST, ReqIF, PDF, JSON, Excel.
- StrictDoc has a web-based user interface which allows viewing and editing the documents and requirements. The changes are written back to .sdoc files.
- The focus of the tool is modeling requirements and specifications documents. Such documents consist of multiple statements like "system X shall do Y" called requirements.
- The requirements can be linked together to form the relationships, such as "parent-child". From these connections, many useful features, such as Requirements Traceability and Documentation Coverage, can be derived.
- Requirements to source files traceability (experimental). See *Traceability between requirements and source code*.
- Custom grammar and custom fields support. The StrictDoc's grammar can be extended to support arbitrary special fields, such as `PRIORITY`, `OWNER`, or even more specialized fields, such as `Automotive Safety Integrity Level (ASIL)` or `Verification method`. See *Custom grammars*.
- Good performance of the textX parser and parallelized incremental generation of documents: generation of document trees with up to 2000–3000 requirements into HTML pages stays within a few seconds. From the second run, only changed documents are regenerated. Further performance tuning should be possible.

See also a summary of StrictDoc's existing limitations: *StrictDoc's limitations*.

1.1.1 Contact the developers

Join us in Discord. Here is the invitation link: https://discord.gg/4BAAME9MmG

The author can be also contacted via email.

1.2 Examples

1.2.1 Hello World

“Hello World” example of the SDoc text language:

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: SDOC-HIGH-REQS-MANAGEMENT
TITLE: Requirements management
STATEMENT: StrictDoc shall enable requirements management.
```

Create a file called `hello_world.sdoc` somewhere on your file system and copy the above “Hello World” example text to it. **The file must end with a newline character**.

Open a command-line terminal program supported on your system.

Once you have `strictdoc` installed (see *Installing StrictDoc* below), switch to the directory with the `hello_world.sdoc` file. For example, assuming that the file is now in the `workspace/hello_world` directory in your user folder:

```
cd <your user home directory>/workspace/hello_world
```

Run StrictDoc as follows:

```
strictdoc export hello.sdoc
```

The expected output:

```
$ strictdoc export hello.sdoc
Parallelization: Enabled
Step 'Collect traceability information' start
Step 'Find and read SDoc files' start
Reading SDOC: hello.sdoc ..................................... 0.08s
Step 'Find and read SDoc files' took: 0.09 sec
Step 'Collect traceability information' start
Step 'Collect traceability information' took: 0.01 sec
Step 'Collect traceability information' took: 0.11 sec
Published: StrictDoc ......................................... 0.24s
...
Export completed. Documentation tree can be found at:
.../output/html
```

The HTML output produced so far has been generated statically. Now, start a StrictDoc server from the same directory:

```
strictdoc server .
```

The expected output should contain the following line:

```
INFO:     Uvicorn running on http://127.0.0.1:5111 (Press CTRL+C to quit)
```

Open the URL in the browser and explore the contents of the example.

1.2.2 StrictDoc Examples repository

The strictdoc-examples repository contains a collection of basic examples. Visit the repository and read its README for details.

1.2.3 StrictDoc Templates repository

The strictdoc-templates repository contains a growing collection of templates from the industry standards like DO-178C (aviation) and ECSS-E-ST-40C (space).

1.2.4 Other examples

For a more comprehensive example, check the source file of this documentation which is written using StrictDoc: strictdoc_01_user_guide.sdoc.

- StrictDoc HTML export
- StrictDoc HTML export using Sphinx
- StrictDoc PDF export using Sphinx

1.3 Installing StrictDoc

1.3.1 Requirements

- Python 3.7+
- macOS, Linux or Windows
- Command-line terminal program

Depending on an operating system, a terminal program can be, for example:

- `Terminal` or `iTerm2` on macOS
- `Gnome Terminal` or `konsole` on Linux
- `Terminal` or `PowerShell` on Windows.

A terminal program is required to input all the commands outlined in this user guide.

1.3.2 Installing StrictDoc as a Pip package (recommended way)

```
pip install strictdoc
```

1.3.3 Installing “nightly” StrictDoc as a Pip package

Sometimes, it takes a while before the latest features and fixes reach the stable Pip release. In that case, installing a Pip package from the Git repository directly is possible:

```
pip install -U --pre git+https://github.com/strictdoc-project/strictdoc.git@main
```

1.3.4 Installing StrictDoc into a Docker container

StrictDoc can be invoked inside of a Docker container. To make data available to the Docker container (here: `strictdoc:latest`) as well as to the host system, one needs to mount a volume via `-v` option.

In the host operating system terminal:

```
docker build . -t strictdoc:latest
docker run --name strictdoc --rm -v "$(pwd)/docs:/data" -i -t strictdoc:latest
```

In the container terminal:

```
bash-5.1# strictdoc export .
bash-5.1# exit
```

The documentation resides in `./docs/output/html`.

1.3.5 Installing StrictDoc as a Snap package (not maintained)

This way of installing StrictDoc is not maintained anymore. If you want to use it, refer to the instructions located in `developer/snap/README.md`.

1.4 Running StrictDoc

1.4.1 Static HTML export

The easiest way to see the static HTML export feature in action is to run the *Hello World* example.

The `export` command is the main producer of documentation. The native export format of StrictDoc is HTML. The `export` command supports a number of parameters, including the option for selecting export formats (HTML, RST, Excel, etc.). The options can be explored with the `--help` command.

```
strictdoc export --help
```

1.4.2 Web server

StrictDoc supports a web-based user interface. The StrictDoc web server is launched via the `server` command which accepts a path to a documentation tree as a parameter.

```
strictdoc server .
```

The `server` command accepts a number of options. To explore the options, run:

```
strictdoc server --help
```

Note: The implementation of the web interface is work-in-progress. See *Limitations of web user interface* for an overview of the existing limitations.

1.4.3 Security considerations

Warning: TL;DR: StrictDoc's web server is not yet hardened against unsafe use. Making StrictDoc safe for deployment in public networks is an ongoing effort.

Using StrictDoc's command-line and web interfaces should be more secure if the web server is not deployed on a public network.

Due to current constraints (refer to *Limitations of web user interface*), StrictDoc requires running a server through a command line interface in one window or OS process, and separately committing changes to documents using Git in another window or OS process. Deploying StrictDoc as a shared web server is impractical, as it still requires manual commits to SDoc files via the server's command line using Git. The future development plan for StrictDoc aims to enable its use as a standalone server application, which includes resolving the following security-related issues.

What makes StrictDoc's web server unsafe:

- The web interface is not fully hardened against unsafe inputs, such as malformed strings or files. The web server does not perform comprehensive sanity checks on the size and validity of inputs across all its HTTP endpoints.
- StrictDoc uses the pickle module to cache SDoc files, significantly improving performance. However, the pickle module is not secure. The pickled files are currently stored in the /tmp folder, which poses risks under certain circumstances.
- The security-related properties of the textX/Arpeggio parser are not understood yet. We have opened a request to track this upstream: textX - Security considerations (#422).
- Several uses of regular expressions may be unsafe, some of which have been identified by GitHub's CodeQL analyzer.
- The security aspects of StrictDoc's dependencies have not yet been analyzed.

Known security-related issues are tracked on GitHub, under the "Security" label. As StrictDoc becomes usable without command-line access, all known issues will need to be addressed or acknowledged as known limitations.

We are committed to continuously enhancing the functionality and security of StrictDoc and welcome user feedback and contributions in this area.

1.5 IDE support

StrictDoc language markup (SDoc) can be activated in all IDEs that support the TextMate grammars. When the StrictDoc grammar is integrated into an IDE, the SDoc syntax becomes highlighted just as any other syntax like Markdown, RST, Python, etc.

The TextMate grammars can be defined in either JSON or PLIST formats. The Sublime Text's Syntax is similar to the TextMate grammar but has more capabilities and is no longer backward-compatible with both TextMate's JSON and PLIST grammars.

The following IDEs are known to work:

- Microsoft Visual Studio Code (TextMate JSON)
- JetBrains's PyCharm and WebStorm (TextMate JSON). The other JetBrains IDEs are expected to work too.
- Eclipse (TextMate JSON)
- Sublime Text (Sublime Syntax)

Due to the incompatibilities between these formats, the markup files are provided in separate repositories:

- strictdoc-project/strictdoc.tmLanguage – TextMate grammar files for StrictDoc (JSON)
- strictdoc-project/strictdoc.tmbundle – TextMate grammar files for StrictDoc (PLIST)
- strictdoc-project/strictdoc.sublime-syntax – StrictDoc markup syntax highlighting in Sublime Text.

The instructions for installing the StrictDoc markup can be found in all repositories.

For any other IDE, when possible, it is recommended to use the TextMate JSON format, unless a given IDE is known to only support the TextMate bundle format (`.tmbundle`). The exception is Sublime Text which has its own format.

Note: The TextMate grammar and the Sublime Syntax for StrictDoc only provides syntax highlighting. More advanced features like autocompletion and deep validation of requirements can be only achieved with a dedicated Language Server Protocol (LSP) implementation for StrictDoc. The StrictDoc LSP is on StrictDoc's long-term roadmap, see Enhancement: Language Protocol Server for SDoc text language #577.

1.6 SDoc syntax

StrictDoc defines a special syntax for writing specifications documents. This syntax is called SDoc and it's grammar is encoded with the textX tool.

The grammar is defined using textX language for defining grammars and is located in a single file: grammar.py.

This is how a minimal possible SDoc document looks like:

```
[DOCUMENT]
TITLE: StrictDoc
```

This documentation is written using StrictDoc. Here is the source file: strictdoc_01_user_guide.sdoc.

1.6.1 Document structure

An SDoc document consists of a `[DOCUMENT]` declaration followed by one or many `[REQUIREMENT]` or `[COMPOSITE_REQUIREMENT]` statements which can be grouped into `[SECTION]` blocks.

The following grammatical constructs are currently supported:

- `DOCUMENT`
  - `FREETEXT`
- `REQUIREMENT` and `COMPOSITE_REQUIREMENT`
- `SECTION`
  - `FREETEXT`

Each construct is described in more detail below.

1.6.1.1 Strict rule #1: One empty line between all nodes

StrictDoc's grammar requires each node, such as `[REQUIREMENT]`, `[SECTION]`, etc., to be separated with exactly one empty line from the nodes surrounding it. This rule is valid for all nodes. Absence of an empty line or presence of more than one empty line between two nodes will result in an SDoc parsing error.

1.6.1.2 Strict rule #2: No content is allowed outside of SDoc grammar

StrictDoc's grammar does not allow any content to be written outside of the SDoc grammatical constructs. It is assumed that the critical content shall always be written in form of requirements: `[REQUIREMENT]` and `[COMPOSITE_REQUIREMENT]`. Non-critical content shall be specified using `[FREETEXT]` nodes. By design, the `[FREETEXT]` nodes can be only attached to the `[DOCUMENT]` and `[SECTION]` nodes.

1.6.1.3 Strict rule #3: No empty strings

StrictDoc's grammar does not allow empty strings. This rule is applicable to both single-line and multiline strings and both section fields and requirement fields. A field is either missing or is a non-empty string.

The following patterns are all invalid for single-line fields:

```
[SECTION]
TITLE:

[SECTION]
TITLE: (any number of space characters after colons)

[REQUIREMENT]
STATEMENT:

[REQUIREMENT]
STATEMENT: (any number of space characters after colons)
```

The following patterns are all invalid for multiline fields:

```
[REQUIREMENT]
COMMENT: >>>
<<<

[REQUIREMENT]
COMMENT: >>>
(any number of space characters)
<<<
```

If you need to provide a placeholder for a field that you know has to be filled out soon, add a "TBD" (to be done, by our team) or a "TBC" (to be confirmed with a customer or a supplier) string.

The Project Statistics screen provides metrics for counting the number of TBDs (To Be Determined) and TBCs (To Be Confirmed) in a document, assisting in evaluating the document's maturity. This is a common practice in the regulared industries. See *Project statistics screen* for more details.

1.6.2 Grammar elements

1.6.2.1 Document

The [DOCUMENT] element must always be present in an SDoc document. It is a root of an SDoc document graph.

```
[DOCUMENT]
TITLE: StrictDoc
(newline)
```

The following DOCUMENT fields are allowed:

Table 1: SDoc grammar DOCUMENT fields

Field	Description
TITLE	Title of the document (mandatory)
UID	Unique identifier of the document
VERSION	Current version of the document
CLASSIFICATION	Security classification of the document, e.g. Public, Internal, Restricted, Confidential
REQ_PREFIX	Requirement prefix that should be used for automatic generation of UIDs. See *Automatic assignment of requirements UID*.
ROOT	Defines whether a document is a root object in a traceability graph. A root document is assumed to not have any parent requirements. The project statistics calculation will skip all root document's requirements when calculating the metric `Non-root-level requirements not connected to any parent requirement`.
OPTIONS	Document configuration options

The DOCUMENT declaration must always have a TITLE field. The other fields are optional. The OPTIONS field can be used for specifying the document configuration options. Note: The sequence of the fields is defined by the document's Grammar, i.e. should not be changed.

Finally an optional [FREETEXT] block can be included.

```
[DOCUMENT]
TITLE: StrictDoc
OPTIONS:
  REQUIREMENT_STYLE: Table

[FREETEXT]
StrictDoc is software for writing technical requirements and specifications.
[/FREETEXT]
```

1.6.2.1.1 Document configuration options

The OPTIONS field may have the following attribute fields:

Table 2: SDoc grammar DOCUMENT-OPTIONS fields

Field	Attribute values
ENABLE_MID	False (default), True
MARKUP	RST (default), HTML, Text
AUTO_LEVELS	On (default), Off
REQUIREMENT_STYLE	Inline (default), Table, Zebra
REQUIREMENT_IN_TOC	True (default), False

1.6.2.1.1.1 ENABLE_MID

See *Machine identifiers (MID)*.

1.6.2.1.1.2 MARKUP

The MARKUP option controls which markup renderer will be used. The available options are: RST, HTML and Text. Default is RST.

1.6.2.1.1.3 AUTO_LEVELS

The AUTO_LEVELS option controls StrictDoc's system of automatic numbering of the section levels. The available options are: On / Off. Default is On.

In case of On, the [SECTION].LEVEL fields must be absent or may only contain None to exclude that section from StrictDoc's automatic section numbering. See also *Section without a level*.

In case of Off, all [SECTION].LEVEL fields must be populated.

1.6.2.1.1.4 REQUIREMENT_STYLE

The REQUIREMENT_STYLE option controls whether requirement's elements are displayed inline or as table blocks. The available options are:

- Inline
- Table
- Zebra

Default is Inline.

```
[DOCUMENT]
TITLE: Hello world
OPTIONS:
  REQUIREMENT_STYLE: Inline
```

1.6.2.1.1.5 REQUIREMENT_IN_TOC

The REQUIREMENT_IN_TOC option controls whether requirement's title appear in the table of contents (TOC). The available options are: True / False. Default is True.

```
[DOCUMENT]
TITLE: Hello world
OPTIONS:
  REQUIREMENT_IN_TOC: True
```

1.6.2.2 Requirement

Minimal "Hello World" program with 3 empty requirements:

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]

[REQUIREMENT]

[REQUIREMENT]
```

The following REQUIREMENT fields are supported:

Table 3: SDoc grammar REQUIREMENT fields

Field	Description
UID	Unique identifier of the requirement
LEVEL	Define section/requirement Level numbering
STATUS	Status of the requirement, e.g. Draft, Active, Deleted
TAGS	Tags of the requirement (comma separated AlphaNum words)
TITLE	Title of the requirement
STATEMENT	The statement of the requirement. The field can be single-line or multiline.
RATIONALE	The rationale of the requirement. The field can be single-line or multiline.
COMMENT	Comments to the rationale. The field can be single-line or multiline. Note: Multiple comment fields are possible.
RELATIONS	List of requirement relations. Note: Before StrictDoc v0.0.45, this field was called REFS.

Currently, all [REQUIREMENT]'s fields are optional but most of the time at least the STATEMENT field as well as the TITLE field should be present.

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
TITLE: Requirements management
STATEMENT: StrictDoc shall enable requirements management.
```

1.6.2.2.1 UID

Unique identifier of the requirement.

Observation: Some documents do not use unique identifiers which makes it impossible to trace their requirements to each other. Within StrictDoc's framework, it is assumed that a good requirements document has all of its requirements uniquely identifiable, however, the `UID` field is optional to accommodate for documents without connections between requirements.

StrictDoc does not impose any limitations on the format of a UID. Examples of typical conventions for naming UIDs:

- `REQ-001`, `SCA-001` (scalability), `PERF-001` (performance), etc.
- `cES1008`, `cTBL6000.1` (example from NASA cFS requirements)
- Requirements without a number, e.g. `SDOC-HIGH-DATA-MODEL` (StrictDoc)
- `SAVOIR.OBC.PM.80` (SAVOIR guidelines)

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: SDOC-HIGH-DATA-MODEL
STATEMENT: STATEMENT: StrictDoc shall be based on a well-defined data model.
```

1.6.2.2.2 Level

Also a `[REQUIREMENT]` can have no section level attached to it. To enable this behavior, the field `LEVEL` has to be set to `None`.

1.6.2.2.3 Status

Defines the current status of the `[REQUIREMENT]`, e.g. `Draft`, `Active`, `Deleted`.

1.6.2.2.4 Tags

Allows to add tags to a `[REQUIREMENT]`. Tags are a comma separated list of single words. Only Alphanumeric tags (a-z, A-Z, 0-9 and underscore) are supported.

1.6.2.2.5 Relations (previously REFS)

The `RELATIONS` field is used to connect requirements to each other:

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: REQ-001
STATEMENT: StrictDoc shall enable requirements management.

[REQUIREMENT]
UID: REQ-002
TITLE: Requirement #2's title
STATEMENT: Requirement #2 statement
```

(continues on next page)

(continued from previous page)

```
RELATIONS:
- TYPE: Parent
  VALUE: REQ-001
- TYPE: File
  VALUE: /full/path/file.py
```

The supported relation types are: `Parent`, `Child`, and `File`. To be used in a requirement, the relations must be first registered in the document grammar. The default grammar defines `Parent` and `File` relation. See *Relations* for more details.

The `RELATIONS` must be the last field of a requirement. For `TYPE: Parent` and `TYPE: Child` relations, the `VALUE` attribute contains a parent/child's requirement `UID`. A requirement may reference multiple parent or child requirements by adding multiple TYPE/VALUE items. Defining circular references e.g. `Req-A` ⇒ `Req-B` ⇒ `Reg-C` ⇒ `Req-A` results in validation errors and must be avoided.

The `TYPE: File`-`VALUE` attribute contains a filename referencing the implementation of (parts of) this requirement. A requirement may add multiple file references requirements by adding multiple `TYPE: File`-`VALUE` items.

Note: The `TYPE: Parent` and `TYPE: Child` are currently the only fully supported types of connection. Linking requirements to files is still experimental (see also *Traceability between requirements and source code*).

Note: In most requirements projects, only the Parent relations should be used, possibly with roles. The Child relation should be used only in specific cases. See *Parent vs Child relations* for more details.

Note: In the near future, adding information about external references (e.g. company policy documents, technical specifications, regulatory requirements, etc.) is planned.

Note: By design, StrictDoc will only show parent or child links if both requirements connected with a reference have `UID` defined.

1.6.2.2.5.1 Requirement relation roles

A requirement relation can be specialized with a role. The role must be registered in the document grammar, see *Relations*.

```
[DOCUMENT]
TITLE: Example

[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  ...
  RELATIONS:
  - TYPE: Parent
    ROLE: Refines

[REQUIREMENT]
UID: REQ-2
TITLE: Requirement title
STATEMENT: >>>
Requirement statement.
<<<
RELATIONS:
- TYPE: Parent
  VALUE: REQ-1
  ROLE: Refines
```

1.6.2.2.6 Title

The title of the requirement. Every requirement should have its TITLE field specified.

Observation: Many real-world documents have requirements with statements and titles but some documents only use statements without title in which case their UID becomes their TITLE and vice versa. Example:

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: REQ-001
STATEMENT: StrictDoc shall enable requirements management.
```

1.6.2.2.7 Statement

The statement of the requirement. The field can be single-line or multiline. Every requirement shall have its STATEMENT field specified.

1.6.2.2.8 Rationale

A requirement should have a RATIONALE field that explains/justifies why the requirement exists. Like comments, the rationale field can be single-line or multiline.

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: REQ-001
STATEMENT: StrictDoc shall enable requirements management.
COMMENT: Clarify the meaning or give additional information here.
RATIONALE: The presence of the REQ-001 is justified.
```

1.6.2.2.9 Comment

A requirement can have one or more comments explaining the requirement. The comments can be single-line or multiline.

```
[DOCUMENT]
TITLE: StrictDoc

[REQUIREMENT]
UID: REQ-001
STATEMENT: StrictDoc shall enable requirements management.
COMMENT: Clarify the meaning or give additional information here.
COMMENT: >>>
This is a multiline comment.

The content is split via \n\n.

Each line is rendered as a separate paragraph.
<<<
```

1.6.2.3 Section

The `[SECTION]` element is used for creating document chapters and grouping requirements into logical groups. It is equivalent to the use of #, ##, ###, etc., in Markdown and ====, `----`, ~~~~ in RST.

```
[DOCUMENT]
TITLE: StrictDoc

[SECTION]
TITLE: High-level requirements

[REQUIREMENT]
UID: HIGH-001
STATEMENT: ...

[/SECTION]

[SECTION]
TITLE: Implementation requirements

[REQUIREMENT]
UID: IMPL-001
STATEMENT: ...

[/SECTION]
```

1.6.2.3.1 Nesting sections

Sections can be nested within each other.

```
[DOCUMENT]
TITLE: StrictDoc

[SECTION]
TITLE: Chapter

[SECTION]
TITLE: Subchapter

[REQUIREMENT]
STATEMENT: ...

[/SECTION]

[/SECTION]
```

StrictDoc creates section numbers automatically. In the example above, the sections will have their titles numbered accordingly: `1 Chapter` and `1.1 Subchapter`.

1.6.2.3.2 Free text

A section can have a block of [FREETEXT] connected to it:

```
[DOCUMENT]
TITLE: StrictDoc

[SECTION]
TITLE: Free text

[FREETEXT]
A sections can have a block of ``[FREETEXT]`` connected to it:

...
[/FREETEXT]

[/SECTION]
```

According to the Strict Rule #2, arbitrary content cannot be written outside of StrictDoc's grammar structure. [SECTION] / [FREETEXT] is therefore a designated grammar element for writing free text content.

Note: Free text can also be called "nonnormative" or "informative" text because it does not contribute anything to the traceability information of the document. The nonnormative text is there to give a context to the reader and help with the conceptual understanding of the information. If a certain information influences or is influenced by existing requirements, it has to be promoted to the requirement level: the information has to be broken down into atomic [REQUIREMENT] statements and get connected to the other requirement statements in the document.

1.6.2.3.3 Section without a level

A section can have no level attached to it. To enable this behavior, the field LEVEL has to be set to None.

```
[DOCUMENT]
TITLE: Hello world doc

[SECTION]
TITLE: Section 1

[/SECTION]

[SECTION]
LEVEL: None
TITLE: Out-of-band Section

[/SECTION]

[SECTION]
TITLE: Section 2

[/SECTION]
```

The section with no level will be skipped by StrictDoc's system of automatic numbering of the section levels (1, 1.1, 1.2, 2, ...).

The behavior of the LEVEL: None option is recursive. If a parent section has its LEVEL set to None, all its subsections' and requirements' levels are set to LEVEL: None by StrictDoc automatically.

1.6.2.4 Composing documents from other documents

Note: The composable documents is an early feature with only 50%+ of the implementation complete. See Epic: UI: Composable documents.

StrictDoc `.sdoc` files can be built-up from including other documents where a document can be included to no more than one including document.

The `[DOCUMENT_FROM_FILE]` element can be used anywhere body elements can be used (e.g. `[SECTION]`, `[REQUIREMENT`, `[COMPOSITE_REQUIREMENT]` etc.) and will evaluate by inserting its contents from the file referenced by its `FILE:` property where it was used in the parent document. The files included must be proper SDoc documents and have a usual `.sdoc` extension.

Here is an example pair of files similar to examples above. First the `.sdoc` file has a `[DOCUMENT_FROM_FILE]` that references the latter file.

```
[DOCUMENT]
TITLE: StrictDoc

[FREETEXT]
...
[/FREETEXT]

[DOCUMENT_FROM_FILE]
FILE: include.sdoc

[REQUIREMENT]
```

Then the referenced file, `include.sdoc`:

```
[DOCUMENT]
TITLE: Section ABC

[REQUIREMENT]

[SECTION]
TITLE: Sub section
[/SECTION]

[COMPOSITE_REQUIREMENT]

[REQUIREMENT]

[/COMPOSITE_REQUIREMENT]
```

Which will resolve to the following document after inclusion:

```
[DOCUMENT]
TITLE: StrictDoc

[FREETEXT]
...
[/FREETEXT]

[SECTION]
TITLE: Section ABC

[REQUIREMENT]

[SECTION]
```

(continues on next page)

(continued from previous page)

```
TITLE: Sub section
[/SECTION]

[COMPOSITE_REQUIREMENT]

[REQUIREMENT]

[/COMPOSITE_REQUIREMENT]

[/SECTION]

[REQUIREMENT]
```

Note: The Composable Documents feature belongs to the list of features that may be less portable when it comes to interfacing with other tools. See *Portability considerations*.

1.6.2.5 Composite requirement

Note: The composite requirements feature shows promise, but it has not yet attracted significant demand from both the core developers of StrictDoc and its users. While the use of composite requirements via the command line is implemented and supported, the web interface does not currently offer this support. Experience has shown that composite requirements can often be represented as a combination of sections and standard requirements. If there is a compelling use case for full support of composite requirements, please reach out to the developers.

A `[COMPOSITE_REQUIREMENT]` is a requirement that combines requirement properties of a `[REQUIREMENT]` element and grouping features of a `[SECTION]` element. This element can be useful in lower-level specifications documents where a given section of a document has to describe a single feature and the description requires a one or more levels of nesting. In this case, it might be natural to use a composite requirement that is tightly connected to a few related sub-requirements.

```
[COMPOSITE_REQUIREMENT]
STATEMENT: Statement

[REQUIREMENT]
STATEMENT: Substatement #1

[REQUIREMENT]
STATEMENT: Substatement #2

[REQUIREMENT]
STATEMENT: Substatement #3

[/COMPOSITE_REQUIREMENT]
```

Special feature of `[COMPOSITE_REQUIREMENT]`: like `[SECTION]` element, the `[COMPOSITE_REQUIREMENT]` elements can be nested within each other. However, `[COMPOSITE_REQUIREMENT]` cannot nest sections.

Note: Composite requirements should not be used in every document. Most often, a more basic combination of nested `[SECTION]` and `[REQUIREMENT]` elements should do the job.

1.6.3 Machine identifiers (MID)

StrictDoc supports the automatic generation of machine identifiers (MIDs). This optional feature can be enabled individually for each document through the document-level ENABLE_MID config option:

```
[DOCUMENT]
TITLE: Hello World!
OPTIONS:
  ENABLE_MID: True
```

When the ENABLE_MID option is enabled, StrictDoc automatically generates MID fields whenever the document is written back to the file system. On the web server, MIDs are generated automatically when a document, section, or requirement is saved. In the command-line interface, the generation of MID can be initiated with a passthrough command. Executing strictdoc passthrough on a document with ENABLE_MID: True results in all nodes having auto-generated MIDs. Implementing the ENABLE_MID option on a per-document basis allows for the integration of MID-enabled documents alongside third-party documents where the MID feature may not be necessary or desired.

Machine identifiers (MIDs) differ from and do not replace unique identifiers (UIDs). A requirement, section, or document node may have both MID and UID fields defined. For example:

```
[REQUIREMENT]
MID: 06ab121d3c0f4d8c94652323b8f735c6
UID: SDOC-SSS-70
STATUS: Active
TITLE: Move nodes between documents
STATEMENT: >>>
The Requirements Tool shall allow moving nodes (sections, requirements) between documents.
<<<
```

Advantages of using machine identifiers:

1. Machine identifiers provide a robust means of identifying documents, sections, requirements, or custom nodes. An MID can uniquely identify a given node, independent of other fields like UID or TITLE.
2. The unique identification of nodes via MIDs enhances the effectiveness of StrictDoc's Diff/Changelog functionality. It allows the algorithm to accurately match requirements, sections, or document nodes, even if they are moved, renamed, or undergo metadata changes.
3. MIDs increase the portability of requirements data. Even when UID naming conventions change or nodes are relocated, the MID continues to uniquely identify the original node.

For larger projects, particularly those with extended maintenance cycles, we strongly recommend activating machine identifiers early in the project lifecycle. This proactive approach ensures robust tracking and management of documentation throughout the project's duration.

1.6.4 Custom grammars

Observation: Different industries have their own types of requirements documents with specialized meta information. Examples: ASIL in the automotive industry or HERITAGE field in some of the requirements documents by NASA.

StrictDoc allows declaration of custom grammars with custom fields that are specific to a particular document.

First, such fields have to be registered on a document level using the [GRAMMAR] field. The following example demonstrates a declaration of a grammar with four fields including a custom VERIFICATION field.

```
[DOCUMENT]
TITLE: How to declare a custom grammar

[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  - TITLE: UID
    TYPE: String
    REQUIRED: True
  - TITLE: VERIFICATION
    TYPE: String
    REQUIRED: True
  - TITLE: TITLE
    TYPE: String
    REQUIRED: True
  - TITLE: STATEMENT
    TYPE: String
    REQUIRED: True
  - TITLE: COMMENT
    TYPE: String
    REQUIRED: True
```

This declaration configures the parser to recognize the declared fields as defined by a user. Declaring a special field as `REQUIRED: True` makes this field mandatory for each and every requirement in the document.

When the fields are registered on the document level, it becomes possible to declare them as the `[REQUIREMENT]` special fields:

```
[REQUIREMENT]
UID: ABC-123
VERIFICATION: Test
STATEMENT: System A shall do B.
COMMENT: Test comment.
```

Note: The order of fields must match the order of their declaration in the grammar.

1.6.4.1 Supported field types

The supported field types are:

Table 4: SDoc grammar field types

Field Type	Description
`String`	Simple String
`SingleChoice`	Enum-like behavior, one choice is possible
`MultipleChoice`	comma-separated words with fixed options
`Tag`	comma-separated list of tags/key words. Only Alphanumeric tags (a-z, A-Z, 0-9 and underscore) are supported.
`Reference`	**DEPRECATED:** comma-separated list with allowed reference types: `ParentReqReference`, `FileReference`. In the newer versions of StrictDoc (0.0.45+), a separate `RELATIONS:` section is used to configure the available relations.

Example:

```
[DOCUMENT]
TITLE: How to declare a custom grammar

[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  - TITLE: UID
    TYPE: String
    REQUIRED: True
  - TITLE: ASIL
    TYPE: SingleChoice(A, B, C, D)
    REQUIRED: True
  - TITLE: VERIFICATION
    TYPE: MultipleChoice(Review, Analysis, Inspection, Test)
    REQUIRED: True
  - TITLE: UNIT
    TYPE: Tag
    REQUIRED: True
  - TITLE: TITLE
    TYPE: String
    REQUIRED: True
  - TITLE: STATEMENT
    TYPE: String
    REQUIRED: True
  - TITLE: COMMENT
    TYPE: String
    REQUIRED: True
    REQUIRED: True
  RELATIONS:
  - Type: Parent
  - Type: File

[FREETEXT]
This document is an example of a simple SDoc custom grammar.
[/FREETEXT]

[REQUIREMENT]
UID: ABC-123
ASIL: A
VERIFICATION: Review, Test
UNIT: OBC, RTU
TITLE: Function B
STATEMENT: System A shall do B.
COMMENT: Test comment.
RELATIONS:
- TYPE: Parent
  VALUE: REQ-001
- TYPE: File
  VALUE: /full/path/file.py
```

1.6.4.2 Reserved fields

While it is possible to declare a grammar with completely custom fields, there is a fixed set of reserved fields that StrictDoc uses for the presentation of the table of contents and the document structure:

Table 5: Reserved fields in SDoc's grammar

Reserved field	Description
UID	Requirement's UID.
RELATIONS (previously REFS)	StrictDoc relies on this field to link requirements together and build traceability information. Note: The REFS field is deprecated and replaced with RELATIONS.
TITLE	Requirement's title. StrictDoc relies on this field to create document structure and table of contents.
STATEMENT	Requirement's statement. StrictDoc presents this field as a long text block.
COMMENT	One or more comments to a requirement.
RATIONALE	The rationale for a requirement. Visually presented in the same way as a comment.

1.6.4.3 Relations

The custom grammar configuration includes the optional `RELATION:` section which specifies the relations a given document supports.

```
[DOCUMENT]
TITLE: Test Doc

[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  - TITLE: STATEMENT
    TYPE: String
    REQUIRED: True
  RELATIONS:
  - TYPE: Parent

[REQUIREMENT]
STATEMENT: >>>
This is a statement.
<<<
RELATIONS:
- TYPE: Parent
  VALUE: ID-001
```

The supported relation types are `Parent`, `Child`, `File`. The Parent/Child relations are valid between requirements, the File relation connects a requirement with a file.

The default grammar relations, when a custom grammar is not specified, are `Parent` and `File`.

1.6.4.3.1 Relation roles

StrictDoc's custom grammar support the configuration of relation roles. The Parent and Child relations can be further specialized with roles, such as Refines, Implements, Verifies, etc.

```
[DOCUMENT]
TITLE: Test Doc

[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  ...
  RELATIONS:
  - TYPE: Parent
    ROLE: Refines
```

With this grammar, StrictDoc will only allow creating requirements that have Parent relations with the `ROLE: Refines` specified. Any other relations will trigger validation errors.

1.6.4.3.2 Parent vs Child relations

TL;DR If there is no compelling reason to use the Child relations, avoid using them.

Most of the technical requirements documents can be modeled with just a Parent relation type. A typical traceability graph for a requirements project is typically child-to-parent, where the higher-level parent requirements are referred to as "Parents" by their child requirements.

For example, in one (parent) document:

```
[REQUIREMENT]
UID: PARENT-1
TITLE: Parent requirement
STATEMENT: >>>
...
<<<
```

Somewhere in another child document:

```
[REQUIREMENT]
UID: CHILD-1
TITLE: Child requirement
STATEMENT: >>>
...
<<<
RELATIONS:
- TYPE: Parent
  VALUE: PARENT-001
```

In some very special cases, it may be desired to also use the Child relations. For example, creating a so-called Compliance Matrix between a standard and a project requirement can use the Child relation to connect both the upper-level standard requirement with a project-level technical requirement:

```
[DOCUMENT]
TITLE: Standard X Compliance Matrix

[GRAMMAR]
ELEMENTS:
...
RELATIONS:
```

(continues on next page)

(continued from previous page)

```
- TYPE: Parent
- TYPE: Child

[REQUIREMENT]
COMPLIANCE: Compliant.
STATEMENT: >>>
This is a compliance statement regarding the Standard X's STANDARD-001 requirement...
<<<
REFS:
- TYPE: Parent
  VALUE: STANDARD-001
- TYPE: Child
  VALUE: PROJECT-001
```

With such a setup, StrictDoc generates the correct traceability graph that will link together the requirements of the PROJECT with the requirements of the STANDARD through the requirements of the compliance matrix.

Another example can be adapting the requirements of the Off-the-Shelf (OTS) project to the higher-level requirements of the user project. An intermediate requirements document can be created that connects the parent requirements of the user project with the immutable child requirements of the OTS project. This intermediate document can link the user requirement with the Parent and the OTS project with a Child link.

Both examples above involve activity called Tailoring when an intermediate document (Compliance Matrix) serves as an interface between two layers of documents.

1.6.4.4 Importing grammar from grammar file

A document grammar can be described in a separate file with an extension `.sgra` and imported to a document. This feature may be useful when multiple documents need to share the same grammar.

Example:

```
[DOCUMENT]
TITLE: Document 1

[GRAMMAR]
IMPORT_FROM_FILE: grammar.sgra

[REQUIREMENT]
TITLE: Requirement title
STATEMENT: >>>
Requirement statement.
<<<
```

A grammar file has an extension `grammar.sgra` and contains a usual grammar declaration which starts with a `[GRAMMAR]` tag.

```
[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  - TITLE: TITLE
    TYPE: String
    REQUIRED: True
  - TITLE: STATEMENT
    TYPE: String
    REQUIRED: True
```

When a [GRAMMAR] is declared with an IMPORT_FROM_FILE line, the grammar from the grammar file becomes the document grammar as if it was declared directly in the document.

Note: Editing of the grammars defined in .sgra files can be only done with a text editor, it is not implemented yet in the editable web interface.

1.6.5 Links

StrictDoc supports creating inline links to document sections and anchors.

1.6.5.1 Section links

When a section has an UID, it is possible to reference this section from any other section's text using a [LINK: <Section UID>] tag.

Example:

The following link references a section: *Links*.

Note: Adding a LINK tag will only work from the section text. In the requirement fields, the LINK tag will not be recognized.

1.6.5.2 Anchors

The [ANCHOR: <anchor uid>, <optional anchor title>] tag creates an anchor that can be referenced from other pages using [LINK <Anchor UID>].

Example:

This is a link to anchor: *Anchor ABC*.

Note: ANCHOR is a block-level tag. It has to be placed in the beginning of a line with a newline break after the tag.

1.6.5.2.1 Anchor example

This section contains an anchor named Anchor ABC.

1.7 Search and filtering

StrictDoc supports the search and filtering of document content.

The web interface includes the Search screen, designed for conducting queries against a document tree. The command-line interface supports filtering of requirements and sections through the export and passthrough commands.

1.7.1 Query engine

The syntax of the search query is inspired by Python, utilizing a fixed grammar that converts search queries into corresponding Python expressions.

Important rules:

- Every query component shall start with `node.`.
- `and` and `or` expressions must be grouped using round brackets.
- Only double quotes are accepted for strings.

Table 6: Query examples

Query	Description
`node.is_requirement`	Find all requirements.
`node.is_section`	Find all sections.
`node.is_root`	Find all requirements or sections from documents with `ROOT: True`. See *Document* for the description of the `ROOT` option.
`(node.is_requirement and "System" in node["TITLE"])`	Find all requirements with a TITLE that equals to "System".
`(node.is_requirement and node.has_parent_requirements)`	Find all requirements which have parent requirements.
`(node.is_requirement and node.has_child_requirements)`	Find all requirements which have child requirements.

1.7.2 Filtering content

Both `export` and `passthrough` command-line interface commands support filtering documentation content with `--filter-requirements` and `--filter-sections` options.

Both options are based on the Query Engine, so the same rules that are valid for Search also apply for filtering. When a filter is applied, only the whitelisted requirements/sections will be exported.

Example:

```
strictdoc export . --filter-requirements '"System" in node["TITLE"]'
```

1.8 Markup

The Restructured Text (reST) markup is the default markup supported by StrictDoc. The reST markup can be written inside all StrictDoc's text blocks, such as `[FREETEXT]`, `STATEMENT`, `COMMENT`, `RATIONALE`.

See the reST syntax documentation for a full reference.

Note: StrictDoc supports a Docutils-subset of RST, not a Sphinx-subset. See *Limitations of RST support by StrictDoc*.

The support of Tex and HTML is planned.

1.8.1 Images

To insert an image into a document, create a folder named `_assets` alongside your document and then place the image file into it.

This is the example of how images are included using the reST syntax:

```
[FREETEXT]
.. image:: _assets/sandbox1.svg
   :alt: Sandbox demo
   :class: image
[/FREETEXT]
```

Note: Currently, it is not possible to upload images via the web user interface. Therefore, you must manually place the image into the `_assets` folder using either the command-line or a file browser.

1.8.2 Mathjax support

StrictDoc can include the MathJax Javascript library to all of the document templates. To activate MathJax, edit the `strictdoc.toml` config file in the root of your repository with documentation content.

```
[project]
title = "My project"

features = [
  "MATHJAX"
]
```

Example of using MathJax:

```
[FREETEXT]
The following fragment will be rendered with MathJax:

.. raw:: latex html
    $$
    \mathbf{\underline{k}}_{\text{a}} =
    \mathbf{\underline{i}}_{\text{a}} \times
    \mathbf{\underline{j}}_{\text{a}}
    $$

[/FREETEXT]
```

See *Selecting features* for the description of other features.

1.9 Export formats

1.9.1 HTML documentation tree by StrictDoc

This is a default export option supported by StrictDoc.

The following command creates an HTML export:

```
strictdoc export docs/ --formats=html --output-dir output-html
```

Example: This documentation is exported by StrictDoc to HTML: StrictDoc HTML export.

The options `--formats=html` and `--output-dir output-html` can be skipped because HTML export is a default export option and the default output folder is `output`.

StrictDoc does not detect .sdoc files in the output folder. This is based on the assumption that StrictDoc should not read anything in the output folder, which is intended for transient output artifacts.

1.9.1.1 Standalone HTML pages

The following command creates a normal HTML export with all pages having their assets embedded into HTML using Data URI / Base64. In the project's `strictdoc.toml` file, specify:

```
[project]

features = [
  "STANDALONE_DOCUMENT_SCREEN"
]
```

The generated document are self-contained HTML pages that can be shared via email as single files. This option might be especially useful if you work with a single document instead of a documentation tree with multiple documents.

1.9.2 HTML export via Sphinx

The following command creates an RST export:

```
strictdoc export YourDoc.sdoc --formats=rst --output-dir output
```

The created RST files can be copied to a project created using Sphinx, see Getting Started with Sphinx.

```
cp -v output/YourDoc.rst docs/sphinx/source/
cd docs/sphinx && make html
```

StrictDoc's own Sphinx/HTML documentation is generated this way, see the Invoke task: invoke sphinx.

1.9.3 PDF export via Sphinx/LaTeX

The following command creates an RST export:

```
strictdoc export YourDoc.sdoc --formats=rst --output-dir output
```

The created RST files can be copied to a project created using Sphinx, see Getting Started with Sphinx.

```
cp -v output/YourDoc.rst docs/sphinx/source/
cd docs/sphinx && make pdf
```

StrictDoc's own Sphinx/PDF documentation is generated this way, see the Invoke task: invoke sphinx.

1.9.4 JSON

The following command creates a JSON export:

```
strictdoc export YourDoc.sdoc --formats=json --output-dir output/
```

The structure of the exported JSON mostly mirrors the structure of the underlying SDoc objects that represent the project tree, documents, sections, requirements, and other nodes.

When the exported documents are included to other documents using the *Composing documents from other documents* feature, the JSON export does not include the included documents but only the including documents with the included content. This can be changed by adding the `--included-documents` option.

1.10 Manage project tree

1.10.1 Automatic assignment of requirements UID

To assign requirement UIDs automatically:

```
strictdoc manage auto-uid <path-to-project-tree>
```

The command goes over all requirements in the project tree and assigns missing UIDs automatically. The project tree is mutated in-place.

By default, the assignment happens based on the requirement mask `REQ-`, so the requirements will get the UIDs of `REQ-001`, `REQ-002`, ...

If a document-level or a section-level requirement mask is provided, the UIDs will be generated based on that mask.

A document-level requirement mask:

```
[DOCUMENT]
TITLE: Hello world doc
REQ_PREFIX: MYDOC-
```

A section-level requirement mask:

```
[SECTION]
TITLE: Section 2.
REQ_PREFIX: LEVEL2-REQ-
```

1.11 Traceability between requirements and source code

Note: This feature is experimental, the documentation is incomplete.

StrictDoc allows connecting requirements to source code files. Two types of links are supported:

1) A basic link where a requirement links to a whole file.

```
[REQUIREMENT]
UID: REQ-001
RELATIONS:
- TYPE: File
  VALUE: file.py
TITLE: File reference
STATEMENT: This requirement references the file.
```

2) A range-based link where a requirement links to a file and additionally in the file, there is a reverse link that connects a source range back to the requirement:

The requirement declaration contains a reference of the type `File`:

```
[REQUIREMENT]
UID: REQ-002
RELATIONS:
- TYPE: File
  VALUE: file.py
TITLE: Range file reference
STATEMENT: This requirement references the file.py file.
COMMENT: >>>
If the file.py contains a source range that is connected back to this
requirement (REQ-002), the link becomes a link to the source range.
<<<
```

The source file:

```
# @sdoc[REQ-002]
def hello_world():
    print("hello world")
# @sdoc[/REQ-002]
```

To activate the traceability to source files, configure the project config with a dedicated feature:

```
[project]

features = [
  "REQUIREMENT_TO_SOURCE_TRACEABILITY"
]
```

By default, StrictDoc looks for source files in a directory from which the `strictdoc` command is run. This can be changed by using the `source_root_path` project-level option.

See *Project-level options* for more details about the project-level options.

The strictdoc-examples repository contains executable examples including the example of requirements-to-source-code traceability.

1.12 ReqIF support

StrictDoc has an initial support of exporting to and importing from the ReqIF format.

Note: It is not possible to implement a single export/import procedure that works well for all ReqIF XML files produced by various requirements management tools. The export/import workflow is therefore tool-specific. See *ReqIF implementation details* for more details.

Supported formats:

- StrictDoc's "native" export/import between SDoc and ReqIF

Planned formats:

- The format recommended by the ReqIF Implementation Guide that attempts to harmonize the developments of ReqIF by requirements management tools.

1.12.1 Import flow (ReqIF -> SDoc)

```
strictdoc import reqif sdoc input.reqif output.sdoc
```

The command does the following:

1. The ReqIF is parsed from XML file to ReqIF in-memory model using the `reqif` library.
2. The ReqIF in-memory model is converted to SDoc in-memory model. In this case, `sdoc` indicates that the native ReqIF-to-SDoc conversion procedure must be used.
3. The SDoc in-memory model is written to an .sdoc file.

1.12.2 Export flow (SDoc -> ReqIF)

```
strictdoc export --formats=reqif-sdoc %S/input.sdoc
```

The command does the following:

1. The SDoc file is parsed to an SDoc in-memory model.
2. The SDoc in-memory model is converted to a ReqIF in-memory model using the native SDoc-to-ReqIF conversion procedure as indicated by the `reqif-sdoc` argument.
3. The ReqIF in-memory model is unparsed a to ReqIF XML file using `reqif` library.

1.12.3 ReqIF options

The following options are available for ReqIF export/import commands.

`--reqif-multiline-is-xhtml` This option makes StrictDoc to export all multiline fields as XHTML attributes, not as STRING (the default behavior). This is useful for interfacing with tools, such as Polarion, which assume XHTML as the primary format for writing multiline text.

`--reqif-import-markup={RST,HTML,Text}` This option makes StrictDoc import ReqIF to SDoc documents, setting their MARKUP option to the markup value provided. The default value is RST which is the default markup of StrictDoc. When working with other ReqIF tools, very often this option can be set to HTML. It is likely that with this option, the previous option `--reqif-multiline-is-xhtml` should be enabled as well.

`--reqif-enable-mid` This option requires the machine identifiers option to be enabled (see *Machine identifiers (MID)*) and allows all nodes machine identifiers (MID) exported as ReqIF IDENTIFIERs. This option can be useful when the MID/IDENTIFIER stability of document, section, and requirement nodes is critical when doing iterative export/import roundtrips.

1.12.4 ReqIF implementation details

The ReqIF is a standard which is maintained by Object Management Group (OMG). One important feature of the ReqIF standard is that it requires a fixed XML structure but still leaves certain details open to the implementation by the ReqIF and requirements management tools developers. Specifically, each tool may use it own field names and structure to represent requirements and sections/chapters.

In order to accommodate for the differences between ReqIF files produced by various tools, the ReqIF processing is split into two layers:

1) Parsing ReqIF from `.reqif` XML files into ReqIF in-memory tree of Python objects as well as unparsing the ReqIF in-memory tree back to ReqIF XML files is extracted to a separate library: strictdoc-project/reqif.

2) Converting between in-memory trees of SDoc and ReqIF. This layer is part of StrictDoc.

For further overview of the ReqIF format and the `reqif` library's implementation details, refer to strictdoc-project/reqif's documentation.

1.13 Excel support

StrictDoc provides a support for Excel XLS on input and Excel XLSX on output.

On input, the headers of sheet1 are used to put together a custom grammar and the requirements are imported one row per requirement. A best effort is made by the importer to recognize names of headers and map these to strictdoc requirement fields.

Note: A roundtrip "SDoc -> Excel -> SDoc" is not yet supported.

1.13.1 Import flow (Excel XLS -> SDoc)

```
strictdoc import excel basic input.xls output.sdoc
```

The command does the following:

1. The Excel XLS is parsed to SDoc in-memory model using the `xlrd` library.
2. The SDoc in-memory model is written to an .sdoc file.

1.13.2 Export flow (SDoc -> Excel XLSX)

```
strictdoc export --formats=excel --output-dir=Output input.sdoc
```

The command does the following:

1. The SDoc file is parsed to an SDoc in-memory model.
2. The SDoc in-memory model is converted to an Excel XLSX file using the `XlsWriter` library.

For exporting only selected fields:

```
strictdoc export --formats=excel --fields=UID,STATUS --output-dir=Output input.sdoc
```

For exporting a folder with multiple SDoc files, specify a path to a folder or `.` for a current directory:

```
strictdoc export --formats=excel .
```

If the `output-dir` option is not provided, the `output/` folder is the default value.

1.14 Options

1.14.1 Project-level options

StrictDoc supports reading configuration from a TOML file. The file must be called `strictdoc.toml` and shall be stored in the same folder which is provided as a path to the SDoc documents.

For example, `strictdoc export .` will make StrictDoc recognize the config file, if it is stored under the current directory.

1.14.1.1 Project title

This option specifies a project title.

```
[project]
title = "StrictDoc Documentation"
```

1.14.1.2 Path to assets

By default, StrictDoc copies its CSS/JS and other asset files to a folder `_static` in the HTML output directory.

Sometimes, it is desirable to change the folder name. For example, the GitHub Pages static website engine expects the assets to be found in the `assets` folder.

The `html_assets_strictdoc_dir` allows changing the assets folder name:

```
[project]
html_assets_strictdoc_dir = "assets"
```

1.14.1.3 Path to source root

When the `REQUIREMENT_TO_SOURCE_TRACEABILITY` feature is activated, StrictDoc looks for source files in the directory from which the `strictdoc` program is run. This can be changed with the `source_root_path` option.

```
[project]

features = [
  "REQUIREMENT_TO_SOURCE_TRACEABILITY",
]

source_root_path = "source_root/"
```

The `source_root_path` option supports relative paths, e.g. `../source_root/`.

1.14.1.4 Include/exclude document paths

Use `include_doc_paths` and `exclude_doc_paths` paths to whitelist/blacklist paths to SDoc documents.

In the following example, StrictDoc will look for all files in the input project directory, except all documents in the `tests/` folder.

```
[project]

include_doc_paths = [
  "**"
]

exclude_doc_paths = [
  "tests/**"
]
```

The behavior of wildcard symbols * and ** is as follows:

- The * expands to any combination of symbols that represent a valid file name, excluding the forward and backward slashes, which limits this wildcard to only match a single directory.

- The ** expands to any combination of valid file name symbols, possibly separated by any number of slashes.

Table 7: Examples of possible filter strings

Example	Description
`docs/* or docs/*.sdoc`	Match all documents found in the `docs/` folder but not in its subdirectories.
`docs/**`	Match all documents found in the `docs/` folder and all its subdirectories.
`**/docs/**`	Match all documents found in the `docs/` folder and all its subdirectories. The `docs/` folder can be a top-level folder or at any level of depth.

1.14.1.5 Include/exclude source files paths

Use `include_source_paths` and `exclude_source_paths` to whitelist/blacklist paths to source files when the traceability between requirements and source files feature is enabled.

```
[project]

features = [
  "REQUIREMENT_TO_SOURCE_TRACEABILITY"
]

include_source_paths = [
  "src/**"
]

exclude_source_paths = [
  "src/tests/**"
]
```

The behavior of the wildcards is the same as for the `include_doc_paths/exclude_doc_paths` options.

1.14.1.6 Selecting features

StrictDoc has optional features and features that are developed with a lower priority.

The feature of exporting the SDoc documents to HTML document view is a core feature and is always enabled. The option `features` allows selecting which additional features should be activated or not.

The following is an example of the default configuration. The same features are active/inactive when the option `features` is not specified.

```
[project]
title = "StrictDoc Documentation"

features = [
  # Stable features that are enabled by default.
  "TABLE_SCREEN",
  "TRACEABILITY_SCREEN",
  "DEEP_TRACEABILITY_SCREEN",

  # Stable features that are disabled by default.
  # "MATHJAX",

  # Experimental features are disabled by default.
  # "REQIF",
```

(continues on next page)

(continued from previous page)

```
  # "HTML2PDF",
  # "PROJECT_STATISTICS_SCREEN",
  # "STANDALONE_DOCUMENT_SCREEN",
  # "TRACEABILITY_MATRIX_SCREEN",
  # "REQUIREMENT_TO_SOURCE_TRACEABILITY"
]
```

See *Experimental features* where the experimental features are outlined.

1.14.1.6.1 Enable all features

To select all available features, stable and experimental, specify ALL_FEATURES.

```
[project]

features = [
  "ALL_FEATURES"
]
```

The advantage of this option is that all feature toggles become activated, and all extra screens and buttons are generated and visible.

The disadvantage is that StrictDoc spends more time rendering extra screens that might not be needed by a particular user.

If ALL_FEATURES is present, all features are activated, regardless of any other features that are also specified or not.

1.14.1.6.2 Disable all features

To disable all features, specify the `features` option but leave it empty:

```
[project]

features = [
  # Nothing specified.
]
```

1.14.1.7 Server configuration

1.14.1.7.1 Host and port

By default, StrictDoc runs the server on `127.0.0.1:5111`.

Use the `[server]` section to configure the host and port as follows.

```
[project]
title = 'Test project with a host "localhost" and a port 5000'

[server]
host = "localhost"
port = 5000
```

1.14.2 Command-line interface options

1.14.2.1 Project title

By default, StrictDoc generates a project tree with a project title “Untitled Project”. To specify the project title use the option `--project-title`.

```
strictdoc export --project-title "My Project" .
```

1.14.2.2 Parallelization

To improve performance for the large document trees (1000+ requirements), StrictDoc parallelizes reading and generation of the documents using process-based parallelization: `multiprocessing.Pool` and `multiprocessing.Queue`.

Parallelization improves performance but can also complicate understanding behavior of the code if something goes wrong.

To disable parallelization use the `--no-parallelization` option:

```
strictdoc export --no-parallelization docs/
```

Note: Currently, only the generation of HTML documents is parallelized, so this option will only have effect on the HTML export. All other export options are run from the main thread. Reading of the SDoc documents is parallelized for all export options and is disabled with this option as well.

1.15 Python API

At present, StrictDoc lacks a documented public Python API. Nevertheless, users can leverage StrictDoc’s internal API to enhance existing functions or create custom import, export, and analysis tools. The architecture of StrictDoc is highly modular, so for each functional block there shall always be a dedicated Python class with a public interface, see *High-level architecture*.

One good example is the `SDWriter` class, which exercises the complete export of the Python data objects to the SDoc format. Since, the SDoc format is the primary data format of StrictDoc, the SDWriter is quite feature-rich in what it does and covers. The `RSTWriter` is less powerful because it does not reflect the full data model, but is probably worth a look as well.

The `strictdoc/backend/reqif` folder contains exporter/importer routines for the ReqIF format. There, the core classes from the Python data model, e.g. SDocNode, Section, Document, Grammar, etc. are created or read from and to the ReqIF format.

The `ManageAutoUIDCommand` class features a good use of all APIs that one may need to read/update/write back a SDoc document tree:

- The `TraceabilityIndex` is created from a project config. The traceability index is the main class for storing the whole traceability graph in Python objects. It has plenty of methods for reading and writing things from the graph.
- The `DocumentUIDAnalyzer` is an example of how the objects are manipulated in memory.
- And finally the final sequence writes the mutated traceability graph back to files using `SDWriter`.

For any custom Python API request, for example, a need to do a more advanced data analysis on SDoc data, open a GitHub issue and your specific issue will be handled.

1.16 Portability considerations

Note: **TL;DR**: The following topic of portability becomes relevant if documentation created with StrictDoc has to be exported to another tool and especially if the other tool has to export the content back to StrictDoc. Writing custom export/import generators may be needed to enable a full interoperability when the less portable features are used.

The portability of documentation, particularly when it involves requirements, shares similarities to the portability of programming languages. StrictDoc has several features that are useful but they can also limit the interoperability of the documentation/requirements when the content is exchanged with other tools.

The following is a list of features that are considered less portable when it comes to interfacing with other tools through the existing export/import interfaces:

- *Composing documents from other documents*. Composing documents from other documents is a useful feature but it may not be directly supported by other tools. When exporting to JSON or ReqIF, StrictDoc by default does not export included documents but only the including documents.
- *Composite requirement*. A Composite Requirement is a useful concept which is partially supported by StrictDoc but it may be supported less by other tools.
- *Section without a level*. Table of contents hierarchy where some nodes do not have TOC levels (or have custom TOC levels) can cause problems when exporting/importing documentation content if an interfacing tool does not support custom TOC nodes.

Note: It is easier to extend StrictDoc to produce a format supported by a given tool than it is to make the other tool export a 100%-identical content back to StrictDoc. If there is a need to interface with a tool X and something is missing in StrictDoc, please reach out to the developers (see *Contact the developers*).

1.17 Experimental features

At any point in time, StrictDoc supports features that are still experimental. These features are either not fully developed or their testing has not been completed yet.

A feature is considered stable when all its known edge cases have been covered and enough users report that they have used and tested this feature.

See also *Selecting features* for general instructions.

1.17.1 Project statistics screen

The project statistics screen displays useful information about a documentation project as well as some requirements-based statistics.

To activate the project statistics screen, add/edit the `strictdoc.toml` config file in the root of your repository with documentation content.

```
[project]
title = "My project"

features = [
```

(continues on next page)

(continued from previous page)

```
  "PROJECT_STATISTICS_SCREEN"
]
```

This feature is not enabled by default because it has not undergone sufficient testing by users. The particular aspect requiring extensive testing is related to StrictDoc's interaction with Git to retrieve git commit information. There remain certain unexamined edge cases and portability concerns, e.g., testing on Windows, testing projects that have no Git version control, calling StrictDoc outside of a project's root folder.

1.17.2 HTML2PDF document generator

StrictDoc offers an experimental feature for converting HTML documents into PDF files. This feature aims to deliver a good PDF printing experience without the necessity of installing more sophisticated printing systems like LaTeX.

There are three methods of PDF printing available:

1. Through the command-line interface with the `strictdoc export --formats=html2pdf ...` command.
2. Within the web interface by clicking the 'Export to PDF' button.
3. Also in the web interface, by navigating to a 'PDF' view of a document and using the browser's built-in Print function.

The first two methods require the Chrome browser to be installed on the user's computer.

The third method, the PDF screen, presents a version of the document that is optimized for browser printing. This approach allows for the creation of neatly formatted PDF documents or directly printed documents. Although this method is compatible with any browser, Chrome is recommended for the best printing results. Unlike Firefox and Safari, Chrome maintains the document's internal hyperlinks in the printed PDF.

To activate the HTML2PDF screen in the web interface, add/edit the `strictdoc.toml` config file in the root of your repository with documentation content.

```
[project]
title = "My project"

features = [
  "HTML2PDF"
]
```

This feature is not enabled by default because the implementation has not been completed yet. The underlying JavaScript library is being improved with respect to how the SDoc HTML content is split between pages, in particular the splitting of HTML <table> tags is being worked out. One feature which is still missing is the ability to generate user-specific front pages with custom meta information.

1.17.3 Mermaid diagramming and charting tool

The Mermaid tool allows to create diagrams inside of StrictDoc/RST markup as follows:

```
[FREETEXT]
.. raw:: html

    <pre class="mermaid">
    graph TD
    A[Enter Chart Definition] --> B(Preview)
    B --> C{decide}
    C --> D[Keep]
    C --> E[Edit Definition]
    E --> B
    D --> F[Save Image and Code]
    F --> B
    </pre>
[/FREETEXT]
```

To activate Mermaid, add/edit the `strictdoc.toml` config file in the root of your repository with documentation content.

```
[project]
title = "My project"

features = [
  "MERMAID"
]
```

This feature is not enabled by default because it has not received enough testing.

1.17.4 Shadow features

At any given moment, StrictDoc may contain one or more features that have been implemented and are supported in the codebase, yet lack documentation.

In most cases, these features are still in their early stages and may not even be documented as experimental features.

The testing of these experimental features is typically done by developers or by selected users who have either requested or expressed interest in a specific feature.

If you happen to stumble upon such a hidden feature, we encourage you to use it and provide bug reports or share your experiences with it. However, please be prepared to encounter various unknown or undefined behaviors in the process.

1.18 StrictDoc's limitations

1.18.1 Limitations of RST support by StrictDoc

StrictDoc uses Docutils for rendering RST to HTML, not Sphinx. The implication is that no Sphinx-specific RST directives are supported. Refer to this issue for the related discussion of the limitations: Unexpected restriction on specific RST directives / compatibility with Breathe Sphinx Plugin #1093.

1.18.2 Limitations of web user interface

The existing implementation of the web user interface is alpha-quality and incomplete. The user interface and the underlying backend implementation are not yet autonomous from the command-line workflow. A user still has to access the command line to run the server and commit the documents to Git manually.

The currently supported workflow for the `server` command must be hybrid:

- In one terminal window: run server.
- In another window: check the changes made by the server in the .sdoc files. Commit the .sdoc files to Git.

Note that currently, StrictDoc server maintains an in-memory state of a documentation tree, and it does not watch over the changes made in the .sdoc files. If you make a change in an `.sdoc` file manually, you have to restart the server in order for your changes to show up in the web user interface.

The following essential features are still missing and will be worked on in the near future:

- Editing of documents with non-string grammar fields is not supported yet. Example: The `SingleChoice` type will not work in the *.sdoc files.
- Adding images to the multiline fields like requirement's `STATEMENT` and section's `FREETEXT`.
- Adding/editing sections with `LEVEL: None`.
- Deleting a document.
- Deleting a section recursively with a correct cleanup of all traceability information.
- A separate screen for editing project settings.
- Editing File-based relations.
- Moving the TOC nodes of a document when it has one or more included documents.
- Editing `.sgra` grammar files.

1.18.2.1 Concurrent use of web user interface

StrictDoc's web user interface does not handle concurrency. If the same requirement/section is edited by two users at the same time, the last write wins.

The measures for handling concurrent use are planned but have been not implemented yet.

1.19 Known issues

This section documents some known issues and non-obvious implementation details.

1.19.1 Exporting document free text to ReqIF and vice versa

ReqIF format does not seem to provide a dedicated convention for a text node to be distinguished from a requirement or a section. StrictDoc implements a workaround: the document's free text is converted to a section with a `ChapterName` field that equals "Abstract". And the other way round: when a ReqIF-to-SDoc converter encounters the first section of a document to be "Abstract", it is converted to a free text.

1.19.2 Running out of semaphores on macOS

This an edge case on macOS: Python crashes in the Parallelizer class when creating an output queue:

```
self.output_queue = multiprocessing.Queue()
```

The fragment of the crash:

```
sl = self._semlock = _multiprocessing.SemLock(
OSError: [Errno 28] No space left on device
```

The existing workaround for this problem is to increase a number of semaphores in the macOS config:

```
sudo sysctl -w kern.posix.sem.max=20000
```

2. F.A.Q.

This document is a list of questions that people ask about StrictDoc.

Missing a question or an answer? Ask it here: *Contact the developers*.

2.1 What is StrictDoc?

StrictDoc is software for writing technical requirements specifications.

StrictDoc is a spare-time open-source project developed by Stanislav Pankevich (@stanislaw) and Maryna Balioura (@mettta) with contributions from the Open Source community.

The project exists since mid-2019.

2.2 Resources about StrictDoc

Talks:

- Application of the SPDX Safety Profile in the Safety Scope of the Zephyr Project. This presentation introduces the SPDX Safety Profile and details its application within the context of the Zephyr Project. There is also a presentation of the Zephyr project's methodology for capturing requirements using StrictDoc and a discussion of the upcoming plans for the integration of SPDX into StrictDoc.

Blog posts:

- Requirement Traceability with All Substance and No Fuss by BUGSENG.

Screencasts/tutorials:

- Automotive SPICE in opensource StrictDoc tool, with System Architecure ideas by Lukasz Juranek.

2.3 Which web server is recommended for StrictDoc documentation?

This question can be answered in two ways. First of all, StrictDoc has its own web server that can be run with `strictdoc server ...`. Refer to the User Guide for further information.

The following suggestions assume that you are looking to using a web server to host the StrictDoc's static HTML export, without using StrictDoc's own web server.

Is your project public or private? If it is public, you could simply use GitHub pages. See how one user of StrictDoc is doing it here: nmfta-repo/vcr-experiment and the resulting static website: nmfta-repo.github.io/vcr-experiment.

A very good alternative to GitHub pages is Read the Docs.

If the project is private, you could use any server that reads HTML files from a folder. For example, Python has an embedded Web Server, see this for example. Also you could try any web server based on Node.JS.

2.4 Is StrictDoc compatible with Sphinx?

StrictDoc is only partially compatible with Sphinx.

StrictDoc does not have Sphinx in its dependency tree, and it does not use any of Sphinx API. Instead, StrictDoc uses Docutils for RST markup support. Using Docutils, StrictDoc can generate SDoc files to RST files.

There are users of StrictDoc who use both StrictDoc and Sphinx. The following workflow is known to work:

- Write some documentation in SDoc files.
- Export an SDoc documentation tree to an RST documentation tree.
- Connect RST documentation tree with a Sphinx setup.
- Run Sphinx against an RST documentation tree, obtain a Sphinx documentation website or LaTex PDF.

There is a GitHub issue Unexpected restriction on specific RST directives / compatibility with Breathe Sphinx Plugin #1093 where a closer bridging between StrictDoc and Sphinx was discussed with no specific and actionable outcome. This comment is especially relevant as well as the one about possible implementation.

2.5 How did the SDoc text language become what it is?

Shortly: The SDoc markup is a hybrid of TOML and YAML with some influence from HTML/XML and ASN.1. Using each of these formats as-is, and also the JSON format, was considered but discarded during the design. The SDoc markup has been pretty stable since its inception but the flexibility of the TextX parser allows easy modifications of the language in case of future evolutions. Any feedback to the current design of the markup language is appreciated.

The main use case for SDoc is to model a structure of a technical document that consists of tens and hundreds of technical requirements. The following high-level requirements for the markup are therefore relevant:

- Encode documents of reasonable size (up to several hundreds or few thousands of A4-printed pages).
- Visualize large blocks of requirements text without too much markup noise.
- Support documents with nested (2-4 levels) or deeply nested structure (detailed technical specifications with up to 9-10 levels of chapter nesting).
- Support multiple fields for requirement meta information which makes a requirement look like “a text with some meta information around it”.

SDoc format is inspired by several formats: TOML, YAML, ASN.1 and HTML/XML.

TOML: Square bracket syntax

From TOML, StrictDoc borrowed the `[]` bracket syntax to create the `[REQUIREMENT]`, `[SECTION]` and other blocks but uses the YAML-like syntax for these blocks’ fields, for example:

```
[REQUIREMENT]
TITLE: Requirement ABC
STATEMENT: The system A shall do B when C.
```

TOML/YAML: Arrays/dictionaries

StrictDoc has a rudimentary support of arrays and dictionaries. For example, the syntax for defining the document's `[GRAMMAR]` resembles what would look like an array of records in YAML:

```
[GRAMMAR]
ELEMENTS:
- TAG: REQUIREMENT
  FIELDS:
  - TITLE: UID
    TYPE: String
    REQUIRED: True
  - TITLE: LEVEL
    TYPE: String
    REQUIRED: False
```

Capitalization of reserved keywords from ASN.1

From ASN.1, StrictDoc borrows the idea of having all reserved fields capitalized. This helps to visually distinguish between the grammar content and user content.

Nested sections

From HTML, the idea of opening and closing tags is taken to avoid any nesting that would otherwise be required to support the deeply nested documents with up to 6 or 8 levels, e.g., 1.1.1.1.1.1.1...

```
[SECTION]
TITLE: Section 1

[SECTION]
TITLE: Section 1.1

...

[/SECTION]

[/SECTION]
```

Taking HTML or XML as-is didn't seem like a good option because of the heavy visual noise that is produced around the actual content by the surrounding tags.

Multiline strings

The support of multiline strings is arranged by a custom solution which helps to avoid any nesting of multiline text as well as to visually indicate the start and end parts of the multiline string in a visually unambiguous way. This is how the multiline string is declared:

```
[REQUIREMENT]
TITLE: Requirement ABC
STATEMENT: >>>
The multiline requirement statement
without any nesting.
>>>
```

Discarded options

Taking TOML or YAML as-is didn't seem like a good option because these formats are designed to be used for configuration files or data serialization and not for large documents with hundreds of requirements. The most obvious problems for reusing either of TOML or YAML directly would have been with encoding the deeply nested documents and supporting readable and non-nested multiline strings.

2.6 How StrictDoc compares to other tools?

2.6.1 Doorstop

The StrictDoc project is a close successor of another project called Doorstop.

> Doorstop is a requirements management tool that facilitates the storage of textual requirements alongside source code in version control.

The author of Doorstop has published a paper about Doorstop where the rationale behind text-based requirements management is provided.

The first version of StrictDoc had started as a fork of the Doorstop project. However, after a while, the StrictDoc was started from scratch as a separate project. At this point, StrictDoc and Doorstop do not share any code but StrictDoc still shares with Doorstop their common underlying design principles:

- Both Doorstop and StrictDoc are written using Python. Both are pip packages which are easy-to-install.
- Both Doorstop and StrictDoc provide a command-line interface.
- Both Doorstop and StrictDoc use text files for requirements management.
- Both Doorstop and StrictDoc encourage collocation of code and documentation. When documentation is hosted close to code it has less chances of diverging from the actual implementation or becoming outdated.
- As the free and open source projects, both Doorstop and StrictDoc seem to struggle to find resources for development of specialized GUI interfaces this is why both tools give a preference to supporting exporting documentation pages to HTML format as the primary export feature.

StrictDoc differs from Doorstop in a number of aspects:

- Doorstop stores requirements in YAML files, one separate file per requirement (example). The document in Doorstop is assembled from the requirements files into a single logical document during the document generation process. StrictDoc's documentation unit is one document stored in an .sdoc file. Such a document can have multiple requirements grouped by sections.
- In YAML files, Doorstop stores requirements properties such as `normative: true` or `level: 2.3` for which Doorstop provides validations. Such a design decision, in fact, assumes an existence of implicitly-defined grammar which is encoded "ad-hoc" in the parsing and validation rules of Doorstop. StrictDoc takes a different approach and defines its grammar explicitly using a tool for creating Domain-Specific Languages called textX. TextX support allows StrictDoc to encode a strict type-safe grammar in a single grammar file that StrictDoc uses to parse the documentation files using the parsing capabilities provided by textX out of the box.

The roadmap of StrictDoc contains a work item for supporting the export/import to/from Doorstop format.

2.6.2 Sphinx

Both Sphinx and StrictDoc are both documentation generators but StrictDoc is at a higher level of abstraction: StrictDoc's specialization is requirements and specifications documents. StrictDoc can generate documentation to a number of formats including HTML format as well as the RST format which is a default input format for Sphinx. A two stage generation is therefore possible: StrictDoc generates RST documentation which then can be generated to HTML, PDF, and other formats using Sphinx.

If you are reading this documentation at https://strictdoc.readthedocs.io/en/latest then you are already looking at the example: this documentation stored in strictdoc_02_faq is converted to RST format by StrictDoc which is further converted to the HTML website by readthedocs which uses

Sphinx under the hood. The `StrictDoc -> RST -> Sphinx -> PDF` example is also generated using readthedocs: StrictDoc.

2.6.3 Sphinx-Needs

Sphinx-Needs is a text-based requirements management system based on Sphinx. It is implemented as a Sphinx extension which extends the reStructuredText (RST) markup language with an additional syntax for writing requirements documents.

Sphinx-Needs was a great source of inspiration for the second version of StrictDoc which was first implemented as a Sphinx extension and then as a more independent library on top of docutils that Sphinx uses for the underlying RST syntax processing work.

The similarities between Sphinx-Needs and StrictDoc:

- In contrast to Doorstop, both Sphinx-Needs and StrictDoc do not split a document into many small files, one file per single requirement (see discussion doorstop#401). Both tools follow the "file per document" approach.
- Sphinx-Needs has a well-developed language based on custom RST directives, such as `req::`, `spec::`, `needtable::`, etc. The RST document is parsed by Sphinx/docutils into RST abstract syntax tree (AST) which allows creating an object graph out for the documents and their requirements from the RST document. StrictDoc uses textX for building an AST from a SDoc document. Essentially, both Sphinx-Needs and StrictDoc work in a similar way but use different markup languages and tooling for the job.

The difference between Sphinx-Needs and StrictDoc:

- RST tooling provided by Sphinx/docutils is very powerful, yet it can also be rather limiting. The RST syntax and underlying docutils tooling do not allow much flexibility needed for creating a language for defining requirements using a custom and explicit grammar, a feature that became a cornerstone of StrictDoc. This was a major reason why the third generation of StrictDoc started with a migration from docutils to textX which is a dedicated tool for creating custom Domain-Specific Languages. After the migration to textX, StrictDoc is no longer restricted to the limitations of the RST document but it is still possible to generate SDoc files to RST using StrictDoc and then further generate RST to HTML/PDF and other formats using Sphinx.
- Sphinx-Needs has an impressive list of config options and features that StrictDoc is missing. Examples: Customizing the look of the requirements, Roles, Services and others.

2.6.4 FRET

FRET is a framework for the elicitation, specification, formalization and understanding of requirements.

- Users enter system requirements in a specialized natural language.
- FRET helps understanding and review of semantics by utilizing a variety of forms for each requirement: natural language description, formal mathematical logics, and diagrams.
- Requirements can be defined in a hierarchical fashion and can be exported in a variety of forms to be used by analysis tools.

FRET has an impressive list of Publications.

FRET's user interface is built with Electron.

The detailed comparison is coming.

2.7 How long has the StrictDoc project been around?

The first StrictDoc commit dates back to `2019-08-10`. A short development chronology of StrictDoc is as follows:

2019 – July – August

StrictDoc is a result of several attempts to find a solution for working with text-based requirements. The first version of StrictDoc had started as a fork of the Doorstop project. However, after a while, StrictDoc was started from scratch as a separate project.

2019 – August

StrictDoc, first generation, the first commit dates to `2019-08-10`. Markdown-based C++ program. Custom requirements metadata in YAML.

2020 – January

StrictDoc, second generation: RST/Sphinx-based Python program. Using Sphinx extensions to manage meta information.

2020 – May

The current StrictDoc repository was created on GitHub: the first commit dates back `2020-05-14`. The code still uses RST for parsing requirements meta information and PySide for GUI.

The result of these efforts was the realization that a text-based requirements and specifications management tool could be built on top of a domain-specific language (DSL) created specifically for the purpose of writing requirements and specifications documents. Such a language allows explicit definition of a document data model which is called “grammar”.

2020 – July

The custom RST parser was replaced with a TextX-based DSL. Since then, StrictDoc has been using TextX for parsing SDoc files.

2022 – November

The FastAPI/Turbo/Stimulus-based Web interface prototype was created to complement the text-based interface with a graphical user interface (GUI). When the Web-based GUI is stable, StrictDoc may become useable by non-programmers too.

2.8 Which StrictDoc statistics are available?

Most relevant GitHub statistics:

- Contributors

The pip trends helps to visualize the Pip package download stats. The `reqif` satellite project is included for comparison as well: strictdoc vs reqif.

3. Development Plan

This document presents the goals of the StrictDoc project and describes how the project is developed.

3.1 Project goals

StrictDoc is an open-source tool for writing technical documentation and requirements management. The long-term goal of the project is to provide a capable, open-source platform for creating and managing technical documentation.

Requirements automation

The tool aims to automate the requirements documentation process at various levels.

Target groups

The project targets different user groups including those in software, hardware, systems (systems engineering, electrical, thermal), as well as quality assurance, safety, management, and non-technical areas.

Use on individual computers and cloud

The project is already usable on individual personal computer, the long-term goal is to enable its use on cloud.

Open data

An important feature of StrictDoc is its focus on open data, ensuring ease of data transfer both into and out of the system.

Synergies with other tools

StrictDoc shall be compatible with other software and engineering tools. This includes at least the compatibility with the Python ecosystem, the model-based systems engineering tools, such as Capella, and the formats providing Software Bill of Materials, such as SPDX.

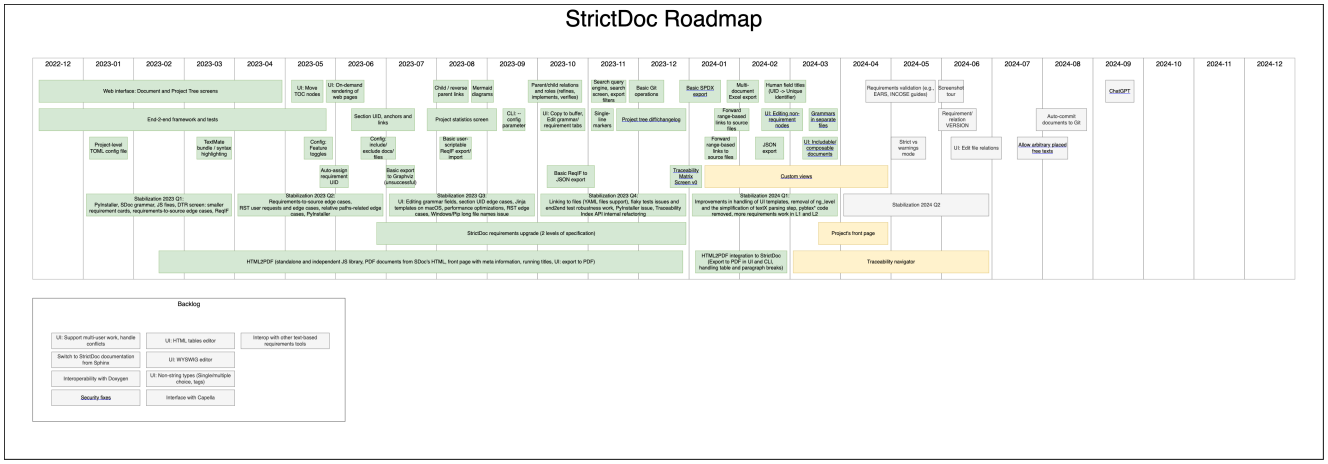
3.2 Project milestones

As an open-source project, StrictDoc is developed without strict deadlines, however there are certain high-level priorities that influence the development. The work is loosely organized in quarters.

Quarter	Planned / accomplished work
2019-Q2	Pre-StrictDoc development in a fork of Doorstop.
2019-Q3	StrictDoc, first prototype. Markdown-based C++ program. Custom requirements metadata in YAML.
2020-Q1	The second prototype of StrictDoc based on RST/Sphinx. Using Sphinx extensions to manage meta information. First integration tests.
2020-Q2	StrictDoc created on GitHub. The code still uses RST for parsing requirements meta information and PySide for GUI.
2020-Q3	The RST parsing is replaced with a TextX-based DSL, new StrictDoc grammar is created. The PySide is replaced with a simple export to HTML using Jinja templates. Export to Sphinx HTML/PDF is introduced.
2020-Q4	Improvements in the styles of HTML/PDF exports. First Table, Traceability, and Deep Traceability screens.
2021-Q1	Excel export. The first implementation for forward and reverse traceability between SDoc and source files.
2021-Q2	Further work on the SDoc-source traceability.
2021-Q3	Further work on the SDoc-source traceability. Tree cycles detection, validations. MathJax support.
2021-Q4	Improvements of the traceability index generation and validation. Initial implementation of ReqIF. First support of custom grammars.
2022-Q1	Further work on ReqIF and custom grammars. Document fragments feature.
2022-Q2	Excel conversion improvements. Improvements of how meta information is displayed in HTML export.
2022-Q3	No work in this quarter.
2022-Q4	Installation using PyInstaller. The first prototype of a Web-based interface. First end-to-end Web tests using SeleniumBase. Improvements of the ReqIF support.
2023-Q1	Improvements of the Web-based interface towards first release. Improvements of the ReqIF support.
2023-Q2	Further stabilization of the Web interface. Graphviz export. RST markup stability improvements. Work on StrictDoc's own requirements traceability.
2023-Q3	Anchors and incoming links. Improvements of the ReqIF interface. Work on StrictDoc's own requirements traceability. Mermaid diagramming tool. Reverse parent / child links. Project statistics screen. Performance improvements.
2023-Q4	Requirements relations and roles. UI improvements and stabilization. Further ReqIF work. Search engine and requirements filters. Project tree Diff/Changelog screens. Basic Git operations.
2024-Q1	TBD

3.2.1 The roadmap diagram

The following diagram contains the work items at the epic and single task levels. This PNG file is exported from a draw.io diagram, where the master version of the roadmap is maintained.



3.3 Verification

StrictDoc has three groups of tests: unit, integration, end-to-end tests.

The unit tests are based on Pylint.

The integration tests are based on the LLVM Integrated Tester and FileCheck.py. These tools are not very common, refer to How to test command-line programs with Python tools: LIT and FileCheck for a good description.

The end-to-end web interface tests are based on SeleniumBase test framework.

3.4 Python baseline

The supported version of Python is kept to be as low as possible. As of 2023-Q4, the currently supported version is Python 3.7.

Ideally, the lowest Python version should only be raised when it is consistently deprecated by the major software platforms like Ubuntu or GitHub Actions.

Another reason for upgrading the minimum Python version is due to the upstream dependencies. As these dependencies stop supporting the older versions of Python, StrictDoc must be upgraded to maintain compatibility. With the existing dependency graph, this happens rather infrequently as most dependencies also maintain the compatibility with the older Python versions.

4. Release Notes

This document maintains a record of all changes to StrictDoc since November 2023. It serves as a user-friendly version of the changelog, complementing the automatically generated, commit-by-commit changelog available here: StrictDoc Changelog.

4.1 0.0.55 (2024-04-28)

The ReqIF export/import feature was extended to support three new command-line options for an improved export/import interfacing with Polarion. See *ReqIF options* for more details.

The Composable Documents feature was extended to support copying assets to the HTML output folder in a redundant way in the case when an included document is stored in a different directory than the parent including document. See https://github.com/strictdoc-project/strictdoc/issues/1777 for the problem definition. Thanks to @Briceus from StrictDoc's Discord channel for reporting this issue.

StrictDoc's caching feature was extended to work around pickling errors when an outdated item is found in a cache. Such issues happen due to the (rare) refactorings in StrictDoc's data model. In this specific case, the previous `FragmentFromFile` Python class was renamed to `DocumentFromFile` and that caused problems when unpickling outdated cached content on a user machine. Thanks to @nashif for reporting this.

4.2 0.0.54 (2024-04-17)

1) Two improvements were made to the Composable Documents feature, when included document's root node is edited in including document:

- If a document is included to another document, now it is possible to edit a title and a free text of the included document.
- It is now possible to add nodes below, above, and inside a root node of an included document. Previously, the UI controls for adding any nodes from the root node were disabled.

2) HTML2PDF feature was updated to support printing UTF8-based documents on Windows.

3) The feature that allows moving TOC (Table of Contents) nodes using drag-and-drop has been enhanced. Now, each TOC element maintains its open or closed state independently of its parent section. Previously, there was some dependency between child and parent TOC nodes, which made quick editing of the TOC more challenging.

4.3 0.0.53 (2024-04-01)

The JSON export algorithm was extended to support composable documents. By default, the included documents are exported only as part of their including documents. To export both the including documents and included documents' standalone SDoc content, the option `--included-documents` option has to be specified with the `export` command.

All code related to pybtex/BibTeX bibliographies has been removed from the StrictDoc project tree. This work was left unfinished for a long time and became unused legacy code over time. See the PR: Remove all BibTeX bibliography-related code and pybtex dependency for more explanation.

4.4 0.0.52 (2024-03-25)

The **Grammar from File** feature has been implemented. Now it is possible to declare a usual StrictDoc `[GRAMMAR]` in a dedicated file with an `.sgra` extension. When a grammar is declared in a separate file, it is possible to share this grammar between several documents. Editing of the grammars defined in `.sgra` files can be only done with a text editor, it is not implemented yet in the editable web interface.

4.5 0.0.51 (2024-03-20)

This is a bugfix release with only one change.

A regression was introduced during recent internal refactoring, resulting in malfunctions on the Search screen when opening search links like “Find all requirements” or “Find all sections.” This release fixes the introduced regression.

4.6 0.0.50 (2024-03-19)

Breaking change: The “Fragments” feature has been replaced by the “Composable documents” feature:

- The command `[FRAGMENT_FROM_FILE]` has been renamed to `[DOCUMENT_FROM_FILE]`.
- Rather than importing section-like fragments, standard SDoc documents can now be included within other SDoc documents.
- The web interface has been updated to support viewing and editing documents both as standalone items and when they are included in other documents.
- Not everything related to the composable documents has been implemented. For example, the ability to drag and drop TOC (Table of Contents) nodes in documents that include other documents. Currently, moving the TOC in documents that include other documents is disabled.
- Further work for the editable web interface can be found here: https://github.com/strictdoc-project/strictdoc/issues/1698.

Other changes:

- The functionality of the HTML2PDF script on Windows has been corrected for scenarios where StrictDoc is operated within a virtual environment. Special thanks to @Timotheous for highlighting this issue.

4.7 0.0.49 (2024-03-11)

The web interface code has been extended to allow editing arbitrary nodes. Previously, only editing the REQUIREMENT type was possible. From now on, it is possible to use the web interface to create custom grammar elements and nodes of corresponding grammar element types.

A basic JSON export feature has been added. Now it is possible to export a StrictDoc project tree to a single JSON file with a structure that mirrors the structure of the SDoc grammar.

Thanks to the work by @dahbar, the SDoc grammar and the web interface have been extended to allow assigning a human title to each field of a grammar element. For example, the `UID` field can be now displayed as `Unique identifier` in the web interface and the static HTML export.

The layout of the PDF document generated by the HTML2PDF conversion process has been improved. Several edge cases, such as the breaks between sentences, have been fixed.

The source file identification mechanism of the requirement-to-source traceability feature has been expanded to locate all source files present in a given source input directory. Previously, it was limited to finding files with specific extensions such as .c, .py, .sdoc, .rst, among others. This restriction, originally implemented for historical reasons, has now been removed. Moreover, StrictDoc has now integrated the `get_lexer_by_name()` function to automatically identify a lexer based on a source file's extension. This enhancement help StrictDoc to offer syntax highlighting tailored specifically to the format of each source file. Previously, StrictDoc's code directly hardcoded only a limited selection of Pygments' lexers. Thanks to @KlfJoat for helping us to prioritize and fix this issue sooner.

The Excel export algorithm was extended to support generating multiple Excel files for documentation tree with requirements that link to each other across documents. The issue manifested itself as `KeyError`. Thanks to @Dynteq for reporting this.

4.8 0.0.48 (2024-01-24)

The requirement-to-source traceability feature was extended to support linking requirements to the RST files.

One more input scenario was handled for the Create Document workflow. When a project config has `include_doc_paths` or `exclude_doc_paths` filters specified, and an input document path contradicts to the provided filters, a validation message is shown.

The Project Statistics screen was extended with the **"Sections without any text" metric**. Now it is possible to visualize which sections are still missing any introduction or description (free text).

The new Machine Identifier (MID) field has been added to StrictDoc's grammar. The automatic generation of MIDs can be activated per-document using the `ENABLE_MID: True` document-level config option. The main driver for this feature is the need of accurate Diff/Changelog results. The new section of the User Guide explains the rationale and the configuration details: *Machine identifiers (MID)*.

The Diff and Changelog screens have been introduced to facilitate a historical comparison of documentation trees. The Diff screen aids in focusing on which document nodes have been altered, while the Changelog functions as a sequential table where changes are displayed as table cells and each cell emphasizes specific details of a particular change.

The Requirements Coverage has been transformed into **the Traceability Matrix** screen. This matrix screen lists all nodes of a documentation graph, along with all their interrelations. The currently generated screen is entirely static. However, future enhancements are planned to include filtering capabilities for the content. The Traceability Matrix feature is disabled by default and has to be activated as `TRACEABILITY_MATRIX_SCREEN` in the strictdoc.toml project config file.

The HTML2PDF feature has now entered the alpha testing phase. This feature enables printing of documents directly from a browser, which can be done either through the "PDF" screen view or by utilizing the "Export to PDF" button. By default, the HTML2PDF feature is disabled. To activate it, you need to indicate the `HTML2PDF` feature in the strictdoc.toml project configuration file.

4.9 0.0.47 (2023-11-20)

A **query search engine** is introduced which allows filtering a documentation tree by queries like `(node.is_requirement and "System" in node["TITLE"])`. Building on the search engine capability, the "Search" screen is introduced in the web interface. Additionally, it is now possible to specify `--filter-requirements <query>` and `filter-sections <query>` when running `export` and `passthrough` commands. The visual design of the project statistics was improved as well as the new design for the search screen has already landed.

The **document option** `ROOT: True/False` was introduced to indicate the root documents in the traceability graph. Currently, this option is only used when printing requirement statistics, where the root nodes are skipped when the metric "requirements without parents" is calculated. The root-level requirements by definition have no parent requirements, they can only be parents to other requirements.

When editing Section, **it is now possible to auto-generate a section UID with a corresponding button** which makes the management of section UIDs much easier.

The **stability and the execution time of the CI end-2-end tests for the web interface has been increased**. The sharding of the end-2-end tests was introduced for all systems: macOS, Linux, and Windows. At the same time, the number of Python versions that are tested by each platform's jobs was reduced to maintain a reasonable total number of build jobs.

The requirement-to-source traceability feature was extended with the so-called **single-line markers**. Now it is possible to reference just a single line in a file by using the `@sdoc(REQ-001)` marker.

Python 3.12 support has been added to the GitHub CI jobs.

The second generation of StrictDoc's requirements received many updates. The new requirements set will be incorporated to the main documentation very soon (estimated time is until the end of 2023). These requirements are maintained in the `drafts/requirements` folder.

The User Guide has been updated to include the **"Security Considerations" chapter**, which provides a warning about unsafe use of StrictDoc if it is deployed to a server on a public network.

5. Contributing to StrictDoc

Contributions to StrictDoc are welcome and appreciated. Presented below is a condensed checklist that summarises the information of the development guide, see *Getting started*.

5.1 Contributor checklist

Before opening your Pull Request, the contributor is encouraged to do the following steps:

1. Run `invoke check` tasks locally. This task calls several lint and test scripts and it is the very task that is run by the GitHub CI process.
2. A contribution that contains changes to the StrictDoc's codebase shall also include tests that exercise the changed behavior. A contribution without any tests is unlikely to be accepted (with the exception of "code climate" changes, see *Python code*).
3. Follow the conventions of the section *Git workflow*. A clean Git history and conventional commit names are expected for every single contribution.
4. If the contribution is not trivial, read through the complete development guide.

5.2 How can I help?

5.2.1 Spread the word

If you like the StrictDoc project and use it in your daily work, there are several things you could do besides contributing Pull Requests:

- Star the StrictDoc repository to show your appreciation of the project.
- Write a blog post or a tutorial about using StrictDoc to achieve some goal.
- Write an email to s.pankevich@gmail.com and mettta@gmail.com and tell us how you are using StrictDoc and which features you are missing. We somewhat lack enough feedback from our users.

5.2.2 ReqIF users

The existing capability of StrictDoc to export/import SDoc to ReqIF is very basic. If you have to deal with ReqIF and you experience errors/crashes when using StrictDoc against ReqIF files, feel free to contribute the anonymized ReqIF files via StrictDoc Issues on GitHub, and we will take care of your specific case.

It is straightforward to create an anonymized version of a ReqIF file. Just reduce your file to the section that causes troubles in import or export and replace all your business-specific titles/texts to some `Lorem ipsum...` boilerplate, see https://www.loremipsum.de/.

5.2.3 TeX / LaTeX / Sphinx experts

The existing template for generating PDF documents using Sphinx looks like this: https://strictdoc.readthedocs.io/_/downloads/en/latest/pdf/. The template is maintained in a separate repository: https://github.com/strictdoc-project/sphinx-latex-reqspec-template and does a good job but could be improved in terms of look and structure used.

If you are an expert and have experience of customizing Sphinx/TeX templates, consider providing feedback or contributing patches.

One extreme way of improving the generated output could be to take the Sphinx template for TeX files and fully customize what Sphinx does to produce a PDF. See https://www.sphinx-doc.org/en/master/latex.html.

6. Developer Guide

This section contains everything that a StrictDoc developer/contributor should know to get the job done.

StrictDoc is based on Python and maintained as any other Python package on GitHub: with linting, tests, and hopefully enough best practice put into the codebase.

The instructions and conventions described below are a summary of what is considered to be the currently preferred development style for StrictDoc.

Any feedback on this development guide is appreciated.

6.1 Getting started

6.1.1 System dependencies

StrictDoc itself mostly depends on other Python Pip packages, so there is nothing special to be installed.

You may need to install `libtidy` if you want to run the integration tests. The HTML markup validation tests depend on libtidy.

On Linux Ubuntu:

```
sudo apt install tidy
```

From the core Python packages, StrictDoc needs Invoke, Tox and TOML:

```
pip install invoke tox toml
```

6.1.1.1 Windows-specific: Long Path support

As reported by a user, Windows Long Path support has to be enabled on a Windows system.

You can find information on how to enable the long paths support at https://pip.pypa.io/warnings/enable-long-paths.

6.1.2 Installing StrictDoc from GitHub (developer mode)

Note: Use this way of installing StrictDoc only if you want to make changes in StrictDoc's source code. Otherwise, install StrictDoc as a normal Pip package by running `pip install strictdoc`.

```
git clone https://github.com/strictdoc-project/strictdoc.git && cd strictdoc
python developer/pip_install_strictdoc_deps.py
python3 strictdoc/cli/main.py
```

The `pip_install_strictdoc_deps.py` installs all dependencies of StrictDoc, but not StrictDoc itself.

6.2 Invoke for development tasks

All development tasks are managed using Invoke in the `tasks.py` file. On macOS and Linux, all tasks run in dedicated virtual environments. On Windows, invoke uses the parent pip environment, which can be a system environment or a user's virtual environment.

Make sure to familiarize yourself with the available developer tasks by running:

```
invoke --list
```

6.3 Main "Check" task

Before doing anything else, run the main check command to make sure that StrictDoc passes all tests on your system:

```
invoke check
```

The check command runs all StrictDoc lint and test tasks with the only exception of end-to-end Web tests that are run with a separate task (see below).

6.4 Python code

- The version of Python is set to be as low as possible given some constraints of StrictDoc's dependencies. Ideally, the lowest Python version should only be raised when it is consistently deprecated by major software platforms like Ubuntu or GitHub Actions.
- All developer tasks are collected in the `tasks.py` which is run by Invoke tool. Run the `invoke --list` command to see the list of available commands.
- Formatting is governed by `black` which reformats the code automatically when the `invoke check` command is run.
  - If a string literal gets too long, it should be split into a multiline literal with each line being a meaningful word or a subsentence.
- For "element is non-None" checks, a full form shall be used, for example: `if foo is not None` instead of `if foo`. This helps to avoid any confusion with all sorts of strings (empty or non-empty `str`, `Optional[str]`) that are used extensively in StrictDoc's codebase. The non-None and non-empty string check shall therefore be as follows: `if foo is not None and len(foo) > 0`. The explicit check also applies to any other kinds of objects besides strings: `if foo is not None` instead of `if foo`. Rationale: `if foo` makes it unclear whether the intention is to check *is not None* or also `len(foo) > 0`.
- For lambdas and short for loops, the recent convention is to add _ to the variables of a for loop or a lambda to visually highlight their temporary use within the current scope which is done to counter the fact that these variables can leak and be used outside of the scope of the loop. Example:

```
for a_, b_ in foo:
    # use a_, b_ within the loop.
```

- The function arguments with the default values shall be avoided. This convention improves the visibility of the function interfaces at the coast of increased verbosity which is the price that StrictDoc development is willing to pay, maintaining the software long-term. The all-explicit function parameters indication is especially useful when the large code refactorings are made.

- StrictDoc has been making a gradual shift towards a stronger type system. Although type annotations haven't been added everywhere in the codebase, it is preferred to include them for all new code that is written.
- If a contribution includes changes in StrictDoc's code, at least the integration-level tests should be added to the `tests/integration`. If the contributed code needs a fine-grained control over the added behavior, adding both unit and integration tests is preferred. The only exception where a contribution can contain no tests is "code climate" which is work which introduces changes in code but no change to the functionality.

6.5 Git workflow

- The preferred Git workflow is "1 commit per 1 PR". If the work truly deserves a sequence of commits, each commit shall be self-contained and pass all checks from the `invoke check` command. The preferred approach: split the work into several independent Pull Requests to simplify the work of the reviewer.
- The branch should be always rebased against the main branch. The `git fetch && git rebase origin/main` is preferred over `git fetch && git merge main`.
- The Git commit message should follow the format:

```
context: description
```

where the context can be a major feature being added or a folder. A form of `context: subcontext: description` is also an option. Typical examples:

`docs: fix links to the grammar.py`

`reqif: native: export/import roundtrip for multiline requirement fields`

`backend/dsl: switch to dynamic fields, with validation`

`Poetry: add filecheck as a dependency`

- Use comma-separated contexts, if the committed work is dedicated to more than one topic. Example:

```
server, UI: update to new requirement styles
```

- When a contribution is simply an improvement of existing code without a change in the functionality, the commit should be named: `Code climate: description`. Example:

```
Code climate: fix all remaining major Pylint warnings
```

6.6 Frontend development

The shortest path to run the server when the StrictDoc's source code is cloned:

```
invoke server
```

6.7 Running End-to-End Web tests

```
invoke test-end2end
```

6.8 Running integration tests

The integration tests are run using Invoke:

```
invoke test-integration
```

The `--focus` parameter can be used to run only selected tests that match a given substring. This helps to avoid running all tests all the time.

```
invoke test-integration --focus <keyword>
```

See How to test command-line programs with Python tools: LIT and FileCheck to learn more about LIT and FileCheck, which enable the StrictDoc integration tests.

6.9 Documentation

- Every change in the functionality or the infrastructure should be documented.
- Every line of documentation shall be no longer than 80 characters. StrictDoc's own documentation has a few exceptions, however, the latest preference is given to 80 characters per line. Unfortunately, until there is automatic support for mixed SDoc/RST content, all long lines shall be edited and split by a contributor manually.
- The `invoke docs` task should be used for re-generating documentation on a developer machine.

6.10 Conventions

- `snake_case` everywhere, no `kebab-case`.
  - This rule applies everywhere where applicable: file and folder names, HTML attributes.
  - Exception: HTML data-attributes and `testid` identifiers.

7. Requirements Tool Specification (L1)

The StrictDoc project is structured around two distinct requirement documents that guide its development:

1. The “Requirements Tool Specification” (this document), which is a higher-level document.
2. The “StrictDoc Requirements Specification,” a more detailed and lower-level document, separate from this one.

This document, the Requirements Tool Specification, delineates the general requirements for an open-source Requirements Tool, with a focus that remains agnostic of the specific implementation details of StrictDoc. Its primary goal is to provide a comprehensive, implementation-neutral overview of what capabilities a Requirements Tool should possess, emphasizing the ‘WHAT’ aspect.

In contrast, the StrictDoc Requirements Specification, the second document, delves deeply into the ‘HOW’ aspect of StrictDoc’s implementation. It builds upon the high-level requirements set out in this document, treating them as parent requirements to guide its detailed development approach.

7.1 Summary of user needs

This section offers an overview of the necessary capabilities of a requirements and documentation management tool.

7.1.1 Free and open source tool

The primary user need and entry point for this Requirements Tool specification is the availability of free, lightweight, yet capable software for creating requirements specifications and other technical documentation. The tool should facilitate the creation of requirements specifications and technical documents, effectively lowering the entry barrier for requirements engineering and technical documentation writing.

It is assumed throughout this document that the requirements tool will be developed as open-source software available at no cost. However, with minor adjustments, these requirements could also apply to commercial requirements tools. See *Licensing and distribution*.

7.1.2 Document types

A requirements tool is very often also a documentation writing and management tool, so, besides the requirements editing functionality, the tool shall be able to accommodate for the variety of documents and templates used in different industries.

The variety comes from:

- Supporting arbitrary documentation structures, e.g., non-nested vs. deeply nested documents.
- Supporting a rich set of visualization mechanisms. Supporting images, diagrams, tables, text markup, etc.
- Supporting custom fields/attributes used by different industries (the criticality levels in various industries, RAIT verification activities in aerospace, status/workflow fields, etc.).

Examples of typical document types include:

- Requirements specification
- Design document / architecture description document

- Interface control document / API reference
- User manual
- Development plan
- Systems engineering plan, management plan
- Standard (e.g., ECSS or ISO 26262).

The documentation/requirements management requirements for a Requirements Tool are specified in the sections *Documentation management* and *Requirements management*.

7.1.2.1 Document structure differences

The table below summarizes various differences observed across industry documents.

Table 1: Table: Requirements and specification document

Criteria	Comments
Structure	Non-nested, nested, deeply nested. A structure of a deeply nested specification document can reach 8-10 levels (e.g., Section 1.2.3.4.5.6.7.8 “ABC”).
Requirements visual presentation	Requirements are often presented as table cells. The cells can be standalone or a whole section or a document can be a structured as a long table with cells. Alternatively, requirements are rather presented as a section header + text.
Unique requirement identifiers (UID)	Some documents provide UIDs for all requirements, some do not. Where the UIDs are missing, the section header numbers are used instead for unique identification. Often, a combination “Number + Title” becomes a referenceable identifier.
Requirement titles	Not all documents provide requirement titles. When the requirement titles are missing, the grouping is may be provided by sections/chapters. When the titles are present, the requirement titles can also be section titles.

7.1.3 Workflows

Besides supporting a variety of document types, a Requirements Tool shall also support and integrate well with multiple documentation and requirements management workflows.

At least the following workflows have been identified to be relevant:

Workflow	Description
Requirements analysis and prototyping	Early phases of the project. Requirements are drafted out quickly. The higher-level picture and coverage of all topics may be more important than specific details.
Requirements compliance, traceability and justification	Demonstrating the correctness of specification, also for contractual purposes. Every aspect shall be traceable.
Requirements implementation	Assisting software engineers in implementing requirements in source code. Making links between requirements and source code implementation units, e.g., files, functions, code fragments. Linking source code implementation units to requirements using special markers left in source code.
Requirements validation/verification workflow (RAIT)	Verifying the correctness of requirements themselves and the validity of the information they communicate.
Configuration management, change management	Time machine and Diff functions. Maintaining requirements baselines (v1, v1.1, v2, etc.).
Reporting	Progress reports, statistics, metrication.
Collaboration on requirements	Supporting multiple users to collaborate on a documentation tree.
Requirements exchange	Integration between distinct projects requirements trees. Example: An embedded software project has its own requirements. The developers want to integrate a requirements subtree of another product that is integrated to the parent project as an off-the-shelf solution.
Formal reviews	Formal review of documentation. Walkthroughs, inspections. Version control of delivered documentation packages. Assessment of progress reports achieved.
Interoperability with industry standards.	Supporting seamless integration between a project documentation tree and applicable standards.

The section *Existing workflows* contains the workflow-related requirements for a Requirements Tool.

7.1.4 Target audience

A Requirements Tool may have different users, each with a different role, experience and background which necessitates the requirements towards usability, installation, and user experience.

The following user groups are preliminarily identified as especially relevant:

Engineering professionals

This group includes:

- Systems engineers
- Requirements engineers
- Assurance experts in quality, safety/security, verification/validation

For these professionals, the Requirements Tool should offer robust functionality that is adequate for complex system design and detailed requirements tracking, ensuring that all aspects of system integrity and compliance are met efficiently.

Management

The focus of this group is more on the progress and compliance aspects. They require high-level overviews and reporting capabilities in the tool, to track project milestones, manage risks, and ensure that the development is adhering to the predefined requirements and industry standards.

Software engineers

For software engineers, the Requirements Tool shall be closely integrated with their software engineering workflow, e.g., it has interfaces with software version control systems, software IDEs, and source code repositories. This integration ensures a seamless transition between requirement specification and software development tasks.

IT/DevOps

This group of users may not work with a Requirements Tool directly but is still an important stakeholder. The Tool shall be easy to install and deploy. It shall be easy to maintain and upgrade the tool, with support for automated updates and compatibility with various IT infrastructures.

General audience

Not all users of a requirements tool must have an engineering background. In fact, there are many projects where non-technical people have to maintain requirements. The Requirements Tool shall be usable without any technical training required, featuring an intuitive user interface and simplified processes for entering and managing requirements.

The requirement sets in the sections *Usability, installation and usage* and *Implementation suggestions* aim to equip the Requirements Tool with sufficient capabilities to support all of the user groups described above.

7.2 Documentation management

This section outlines the requirements towards a Requirements Tool as a documentation tool.

The requirements of this group are dedicated to the core tasks of documentation management:

- Writing, structuring and managing documents
- Complementing documents with meta information
- Versioning documents.

7.2.1 Documents (CRUD)

UID:	SDOC-SSS-3
STATUS:	Active

The Requirements Tool shall provide the CRUD operations for document management:

- Create document
- Read document
- Update document
- Delete document.

RATIONALE:

The CRUD operations are essential operations of document management. They are at the core of a documentation management tool.

Children:

- `[SDOC-SRS-135]` *Free text*
- `[SDOC-SRS-107]` *Create document*
- `[SDOC-SRS-108]` *Delete document*

- [SDOC-SRS-54] *Read document*
- [SDOC-SRS-106] *Update document*

7.2.2 Browsing documentation tree

UID:	SDOC-SSS-91
STATUS:	Active

The Requirements Tool shall provide browsing of the documentation tree.

Children:

- [SDOC-SRS-53] *View project tree*

7.2.3 Documents with nested sections/chapters structure

UID:	SDOC-SSS-51
STATUS:	Active

The Requirements Tool shall allow management of documents with nested sections/chapters structure.

Children:

- [SDOC-SRS-99] *Section model*

7.2.4 Assembling documents from fragments

UID:	SDOC-SSS-52
STATUS:	Active

The Requirements Tool shall allow composing documents from other documents or fragments.

NOTE: If a Requirements Tool implements stores documents in a file system, the composition can be arranged at a file level when a parent document file includes the child fragment files and produces a composite document.

RATIONALE:

Composable documents allow assembling documents from multiple smaller documents which can be standalone documents or document fragments. This feature is particularly useful for managing extensive documents that can be more effectively organized and handled when divided into smaller document sections.

Parents:

- [ZEP-1] *Multiple files / include mechanism*

Children:

- [SDOC-SRS-109] *Composeable document*
- [SDOC-SRS-122] *Importable grammars*

7.2.5 Document meta information (UID, version, authors, signatures, etc)

UID:	SDOC-SSS-53
STATUS:	Active

The Requirements Tool shall support management of document meta information.

Children:

- [SDOC-SRS-110] *Document metadata*

7.2.6 Document versioning

UID:	SDOC-SSS-75
STATUS:	Active

The Requirements Tool shall provide capabilities for document versioning.

Children:

- [SDOC-SRS-110] *Document metadata*
- [SDOC-SRS-111] *Project tree diff*

7.2.7 Text formatting capabilities

UID:	SDOC-SSS-63
STATUS:	Active

The Requirements Tool shall provide “rich text” formatting capabilities which includes but not limited to:

- Headings
- Lists
- Tables
- UML diagrams
- etc.

Parents:

- [ZEP-9] *Text formatting capabilities*

Children:

- [SDOC-SRS-24] *Support RST markup*
- [SDOC-SRS-27] *MathJAX*

7.3 Requirements management

This section outlines the requirements towards a Requirements Tool as a requirements management tool.

The following core aspects of the requirements management are covered:

- Writing and structuring requirements
- Linking requirements with other requirements
- Managing requirement unique identifiers (UID)
- Requirement verification.

7.3.1 Requirements CRUD

UID:	SDOC-SSS-4
STATUS:	Active

The Requirements Tool shall enable the main requirements management operations:

- Create a requirement
- Read / view / browse a requirement
- Update / edit a requirement
- Delete a requirement.

RATIONALE:

The CRUD operations are at the core of the requirements management.

Children:

- `[SDOC-SRS-26]` *Requirement model*
- `[SDOC-SRS-55]` *Edit requirement nodes*

7.3.2 Minimal requirement field set

UID:	SDOC-SSS-61
STATUS:	Active

The Requirements Tool shall support at least the following requirement field set:

- UID
- STATUS
- TITLE
- STATEMENT
- RATIONALE
- COMMENT
- RELATIONS (connections with other requirements).

RATIONALE:

The selection of the minimal set is based on what is common in the industries (e.g., automotive, space, etc).

COMMENT:

The other fields common to each industry can be customized with custom fields handled by other requirements.

Parents:

- [ZEP-10] *Minimal requirement field set*
- [ZEP-14] *Status field*

Children:

- [SDOC-SRS-132] *Requirement model default fields*
- [SDOC-SRS-93] *Default grammar fields*

7.3.3 Custom fields

UID:	SDOC-SSS-62
STATUS:	Active

The requirements tool shall support configuring a requirement item with an arbitrary set of fields.

NOTE: Examples of typical fields include: UID, Title, Statement, Rationale, Comment. Other fields that are used very often are: Status, Tags, Criticality level, Priority, etc.

RATIONALE:

The tool shall not constrain a user in which fields they are able to use for their projects.

Parents:

- [ZEP-3] *Custom fields*

Children:

- [SDOC-SRS-100] *Requirement model fields*
- [SDOC-SRS-21] *Custom grammar / fields*
- [SDOC-SRS-56] *Edit Document grammar*

7.3.4 Structuring requirements in documents

UID:	SDOC-SSS-64
STATUS:	Active

The Requirements Tool shall support structuring requirements in documents.

RATIONALE:

The industry works with requirements documents. The documents have sections/chapters and requirements.

Parents:

- [ZEP-13] *Structuring requirements in documents*

Children:

- [SDOC-SRS-98] *Document model*
- [SDOC-SRS-105] *One document per one SDoc file*

7.3.5 Move requirement nodes within document

UID:	SDOC-SSS-5
STATUS:	Active

The Requirements Tool shall allow moving nodes (sections, requirements) within the containing document.

Children:

- [SDOC-SRS-92] *Move requirement / section nodes within document*

7.3.6 Move nodes between documents

UID:	SDOC-SSS-70
STATUS:	Active

The Requirements Tool shall allow moving nodes (sections, requirements) between documents.

Children:

- [SDOC-SRS-94] *Move requirement / section nodes between documents*

7.3.7 Auto-provision of Requirement UIDs

UID:	SDOC-SSS-6
STATUS:	Active

The Requirements Tool shall provide controls for automatic generation of requirements UIDs.

RATIONALE:

When a document is large, it becomes harder to manage the assignment of the new requirements identifiers by manually exploring which requirement UID has not been assigned yet. The provision of automatically generated UIDs is a convenience feature that improves the user experience significantly.

Parents:

- [ZEP-8] *Unique ID management*

Children:

- [SDOC-SRS-96] *Auto-generate requirements UIDs*
- [SDOC-SRS-85] *Auto-generate requirements UIDs*
- [SDOC-SRS-120] *Auto-completion for requirements UIDs*

7.3.8 Link requirements together

UID:	SDOC-SSS-7
STATUS:	Active

The Requirements Tool shall allow bi-directional linking of requirements nodes together using Parent or Child relations.

RATIONALE:

The requirement ensures a classic capability of a requirement tool: linking requirements together enables bi-directional traceability which helps in understanding how the requirements are related to each other.

Parents:

- `[ZEP-4]` *Links*

Children:

- `[SDOC-SRS-31]` *Requirement relations*
- `[SDOC-SRS-28]` *Traceability index*

7.3.9 Multiple link roles

UID:	SDOC-SSS-8
STATUS:	Active

The Requirements Tool shall support the link roles.

Example of roles for a child-to-parent link: “verifies”, “implements”, “satisfies”, etc.

RATIONALE:

Different industries maintain custom conventions for naming the relations between requirements and other nodes such as tests or other artefacts.

Parents:

- `[ZEP-5]` *Multiple link roles*

Children:

- `[SDOC-SRS-101]` *Requirement relation roles*

7.3.10 Reverse parent links

UID:	SDOC-SSS-71
STATUS:	Active

The Requirements Tool shall support the Reverse Parent relationship.

Children:

- `[SDOC-SRS-102]` *Automatic resolution of reverse relations*

7.3.11 Unique identification of requirements

UID:	SDOC-SSS-89
STATUS:	Active

The Requirements Tool shall provide means for unique identification of every requirement.

RATIONALE:

The requirement ensures a classic capability of a requirement tool:

1) The unique identifiers help the users in identifying the requirements, both when reading a requirements document and when discussing requirements.
2) The unique identifiers are used for linking requirements together. The requirements tool stores the identifiers in a global database and can resolve the links, following the unique identifiers they point to.

Children:

- `[SDOC-SRS-22]` *UID identifier format*
- `[SDOC-SRS-29]` *Uniqueness UID in tree*

7.3.12 Requirements database consistency checks

UID:	SDOC-SSS-47
STATUS:	Active

The Requirements Tool shall provide a validation mechanism that ensures the integrity of requirements and connections between them.

NOTE: Examples of integrity checks:

- Requirements have correct fields.
- Requirements do not form cycles.
- Requirements only link to other requirements as specified in a project configuration.

Children:

- `[SDOC-SRS-30]` *Detect links cycles*
- `[SDOC-SRS-32]` *Link document nodes*

7.3.13 Requirement syntax validation (e.g. EARS)

UID:	SDOC-SSS-57
STATUS:	Active

The Requirements Tool shall provide capabilities for validating requirements according to the EARS syntax.

Children:

- `[SDOC-SRS-116]` *Requirement validation according to EARS syntax*

7.4 Tool configurability

7.4.1 Project-level configuration

UID:	SDOC-SSS-92
STATUS:	Active

The Requirements Tool shall provide a solution for configuring the project-level options.

NOTE: Examples of project-level options:

- Project title.
- Global settings for the Requirements Tool itself.

RATIONALE:

The requirement ensures the configurability of the tool for a specific project.

Children:

- `[SDOC-SRS-37]` *strictdoc.toml file*
- `[SDOC-SRS-39]` *Feature toggles*

7.4.2 Document-level configuration

UID:	SDOC-SSS-93
STATUS:	Active

The Requirements Tool shall provide a solution for configuring the document-level options.

NOTE: Examples of document-level options:

- Document title
- Requirement prefix.
- Other options local to the content and the presentation of a given document.

RATIONALE:

Sometimes, the project-level configuration can be not fine-grained enough. The requirement ensures that the requirements tool allows a configuration on a document level.

Children:

- `[SDOC-SRS-57]` *Edit Document options*

7.5 Performance

This section captures the performance requirements towards a Requirements Tool.

7.5.1 Support large requirements sets

UID:	SDOC-SSS-13
STATUS:	Active

The Requirements Tool shall support requirement trees with at least 10000 requirements.

Children:

- [SDOC-SRS-32] *Link document nodes*
- [SDOC-SRS-1] *Process-based parallelization*
- [SDOC-SRS-95] *Caching of parsed SDoc documents*
- [SDOC-SRS-2] *Incremental generation of documents*
- [SDOC-SRS-3] *Caching of RST fragments*
- [SDOC-SRS-4] *On-demand loading of HTML pages*
- [SDOC-SRS-5] *Precompiled Jinja templates*

7.5.2 Support large project trees

UID:	SDOC-SSS-14
STATUS:	Active

The Requirements Tool shall be able to handle documentation packages of at least 100 documents without significant performance degradation.

Children:

- [SDOC-SRS-32] *Link document nodes*
- [SDOC-SRS-1] *Process-based parallelization*
- [SDOC-SRS-95] *Caching of parsed SDoc documents*
- [SDOC-SRS-2] *Incremental generation of documents*
- [SDOC-SRS-3] *Caching of RST fragments*
- [SDOC-SRS-4] *On-demand loading of HTML pages*
- [SDOC-SRS-5] *Precompiled Jinja templates*

7.6 Data integrity

7.6.1 Data integrity of documentation/requirements

UID:	SDOC-SSS-94
STATUS:	Active

The Requirements Tool shall ensure the integrity of stored documentation and requirements data throughout its lifecycle.

RATIONALE:

The requirement ensures that the tool and the tool's development includes measures for reducing the risk of any data corruption.

COMMENT:

This includes safeguarding against data corruption, loss, and ensuring the reliability of links within the documentation.

Children:

- `[SDOC-SRS-136]` *Identical SDoc content by import/export roundtrip*
- `[SDOC-SRS-127]` *SDoc and Git storage*
- `[SDOC-SRS-19]` *Fixed grammar*
- `[SDOC-SRS-25]` *Type-safe fields*
- `[SDOC-SRS-29]` *Uniqueness UID in tree*
- `[SDOC-SRS-30]` *Detect links cycles*

7.7 Existing workflows

This section captures the requirements towards specific workflows that a Requirements Tool should support as outlined in *Workflows*.

7.7.1 Excel-like viewing and editing of requirements

UID:	SDOC-SSS-73
STATUS:	Active

The Requirements Tool shall provide an Excel-like user interface for viewing and editing requirements.

NOTE: This interface does not have to be the only or a default interface.

RATIONALE:

As recognized by the parent requirement, some requirements-based workflows are naturally easier when the requirements content is presented in a form of a table, as opposed to a document with a nested chapter structure.

Children:

- `[SDOC-SRS-62]` *View TBL screen*

7.7.2 1000-feet view

UID:	SDOC-SSS-56
STATUS:	Active

The Requirements Tool shall provide a “1000-feet view” kind of requirements visualization.

RATIONALE:

Compared to the other visualizations, such a visualization helps to “see the forest for the trees”. Seeing requirements and their sections all at once helps to visualize groups of requirements and better understand the relationships between them.

Children:

- `[SDOC-SRS-90]` *Export to Graphviz/Dot*
- `[SDOC-SRS-113]` *Traceability navigator*

7.7.3 Traceability matrices

UID:	SDOC-SSS-28
STATUS:	Active

The Requirements Tool shall support generation of traceability matrices.

Children:

- `[SDOC-SRS-65]` *View TR screen*
- `[SDOC-SRS-112]` *Traceability matrix*

7.7.4 Compliance matrices

UID:	SDOC-SSS-48
STATUS:	Active

The Requirements Tool shall allow generating a Compliance Matrix document.

Children:

- `[SDOC-SRS-31]` *Requirement relations*
- `[SDOC-SRS-102]` *Automatic resolution of reverse relations*

7.7.5 Requirements coverage

UID:	SDOC-SSS-29
STATUS:	Active

The Requirements Tool shall provide means for getting information about the requirements coverage of a given project.

NOTE: The requirements coverage can be presented in a tabular form or visualized with a set of graphs.

RATIONALE:

The requirements coverage information helps to assess whether all requirements are linked to each other, whether all requirements are connected to implementation, test or other artifacts. Additionally, the requirements coverage information can provide metrics for measuring a project's progress, e.g., "50% of requirements have been traced to the source code". The requirement ensures that the requirements tool provides this feature.

Children:

- `[SDOC-SRS-97]` *Display project statistics*

7.7.6 Progress report

UID:	SDOC-SSS-49
STATUS:	Active

The Requirements Tool shall allow generating a Progress Report document.

NOTE: A progress report document shall include at least the following Key Performance Indicators.

Project-level KPIs:

- Total number of requirements
- Total number of requirements without parent (excluding top-level and derived)
- Total number of TBD/TBC
- Total number of requirements without rationale
- Tags breakdown

Document-level KPIs: the same but per document.

Children:

- `[SDOC-SRS-97]` *Display project statistics*

7.7.7 Change management

UID:	SDOC-SSS-74
STATUS:	Active

The Requirements Tool shall provide capabilities for change management:

- Visualizing changes between project tree versions.
- Visualizing changes between document versions.
- Visualizing the impact that a changed requirement has on a project tree.

Children:

- `[SDOC-SRS-111]` *Project tree diff*
- `[SDOC-SRS-131]` *Update notifications*
- `[SDOC-SRS-117]` *Impact analysis*

7.8 Usability, installation and usage

7.8.1 General usability

UID:	SDOC-SSS-79
STATUS:	Active

The Requirements Tool shall be accessible to a broad spectrum of users.

NOTE: Factors to consider:

- The cost of a tool.
- The easy of installation.
- The availability of a graphical user interface.
- The availability of a programmatic access to the functions of a tool.
- The interoperability of the tool with other tools.

RATIONALE:

A tool that can be used by a large number of people simplifies its adoption and allows more users to work with documentation and requirements.

Children:

- `[SDOC-SRS-50]` *Web interface*
- `[SDOC-SRS-125]` *StrictDoc Python API*
- `[SDOC-SRS-114]` *Web API*

7.8.2 Easy user experience

UID:	SDOC-SSS-80
STATUS:	Active

The Requirements Tool shall provide a smooth user experience.

NOTE: Documentation and requirements management are composite activities that consist of several types of repetitive tasks. A requirements tool user experience should assist in automating these tasks as far as possible and make the overall workflow efficient and precise.

Children:

- `[SDOC-SRS-104]` *SDoc file extension*
- `[SDOC-SRS-50]` *Web interface*
- `[SDOC-SRS-48]` *Preserve generated file names*
- `[SDOC-SRS-96]` *Auto-generate requirements UIDs*
- `[SDOC-SRS-59]` *Buttons to copy text to buffer*
- `[SDOC-SRS-121]` *WYSIWYG editing*
- `[SDOC-SRS-120]` *Auto-completion for requirements UIDs*

7.8.3 Support projects with a large number of users

UID:	SDOC-SSS-81
STATUS:	Active

The Requirements Tool shall be capable of supporting a large number of users.

RATIONALE:

Many documentation and requirements projects involve large groups of people. The requirements tool should not become a bottleneck when a number of users grows.

Children:

- `[SDOC-SRS-123]` *Multi-user editing of documents*

7.8.4 Individual use (home PC)

UID:	SDOC-SSS-82
STATUS:	Active

The Requirements Tool shall be usable on the normal personal computers, e.g., do not require a special cloud deployment.

Children:

- `[SDOC-SRS-87]` *Monolithic application with no microservices*
- `[SDOC-SRS-88]` *No reliance on containerization*
- `[SDOC-SRS-12]` *GitHub*

7.8.5 Server-based deployments (IT-friendly setup)

UID:	SDOC-SSS-83
STATUS:	Active

The Requirements Tool shall be deployable to the network of computers, e.g., provide a server instance.

COMMENT:

Scaling from smaller setups (e.g., Raspberry PI in an office network) to larger in-house and cloud-base installations.

Children:

- `[SDOC-SRS-126]` *Web server*

7.8.6 Requirements database

UID:	SDOC-SSS-84
STATUS:	Active

The Requirements Tool shall store documentation and requirements data in a database.

RATIONALE:

A database allows:

- Persistent storage of documentation/requirements data
- Versioning
- Backups
- Exchange of information and access of the same database by multiple users.

Children:

- `[SDOC-SRS-127]` *SDoc and Git storage*

7.8.7 Programming access via API (Web)

UID:	SDOC-SSS-85
STATUS:	Active

The Requirements Tool shall provide a Web API interface.

RATIONALE:

Besides a direct access to the tool's source code, accessing an API deployed to a server provides additional capabilities for getting and manipulating requirements/documentation content.

Children:

- `[SDOC-SRS-114]` *Web API*

7.8.8 Programming access via API (SDK)

UID:	SDOC-SSS-86
STATUS:	Active

The Requirements Tool shall provide a Software Development Kit (SDK) that allows customization of the Requirements Tool functions.

NOTE: An SDK provides access to the API of the Requirements Tool. Examples of functions that may be used by the users of the tool:

- Custom import/export functions to/from various requirements/documentation formats.
- Implement custom visualization functions.
- Implement integration with other tools.

RATIONALE:

A SDK allows a software engineer to extend the Requirements Tool capabilities.

Children:

- `[SDOC-SRS-125]` *StrictDoc Python API*

7.8.9 Programmatic access to requirements data

UID:	SDOC-SSS-87
STATUS:	Active

The Requirements Tool shall provide programmatic access to requirements data.

RATIONALE:

When the requirements data is accessible by a user directly, it is possible to exchange the data or implement additional scripting procedures.

Children:

- `[SDOC-SRS-127]` *SDoc and Git storage*
- `[SDOC-SRS-125]` *StrictDoc Python API*

7.9 Implementation suggestions

7.9.1 Static HTML export

UID:	SDOC-SSS-30
STATUS:	Active

The Requirements Tool shall support generation of documentation to static HTML.

RATIONALE:

A static HTML export capability enables:

- Viewing requirements in browsers without any additional software.

- Exchanging HTML content as zip between users.
- Publishing HTML content via static website hosting providers (GitHub and GitLab Pages, Read the Docs, Heroku, etc.).

Children:

- `[SDOC-SRS-49]` *Export to static HTML website*

7.9.2 Graphical user interface (GUI)

UID:	SDOC-SSS-31
STATUS:	Active

The Requirements Tool shall provide a graphical user interface.

Children:

- `[SDOC-SRS-50]` *Web interface*

7.9.3 Command-line interface

UID:	SDOC-SSS-32
STATUS:	Active

The Requirements Tool shall provide a command line interface (CLI).

Children:

- `[SDOC-SRS-103]` *Command-line interface*

7.9.4 Web API interface

UID:	SDOC-SSS-68
STATUS:	Active

The Requirements Tool shall provide an API interface.

Children:

- `[SDOC-SRS-114]` *Web API*

7.9.5 Version control (Git)

UID:	SDOC-SSS-33
STATUS:	Active

The Requirements Tool shall support the software version control systems (e.g., Git).

RATIONALE:

- Git allows precise tracking of the changes to the documentation.

- Requirements/documentation content can be release-tagged.
- The “Time machine” function: ability to review the older state of the documentation/requirements tree.

Children:

- [SDOC-SRS-127] *SDoc and Git storage*

7.9.6 Support major operating systems

UID:	SDOC-SSS-67
STATUS:	Active

The Requirements Tool shall support at least the following operating systems:

- Linux
- Windows
- macOS.

Children:

- [SDOC-SRS-9] *Linux*
- [SDOC-SRS-10] *macOS*
- [SDOC-SRS-11] *Windows*

7.9.7 Conservative languages for implementation

UID:	SDOC-SSS-69
STATUS:	Active

The Requirements Tool shall be implemented using the popular programming languages.

NOTE: Examples of the most popular programming languages:

- Java
- C++
- Python
- JavaScript

RATIONALE:

Choosing a less popular programming language can limit the long-term maintainability of the tool.

COMMENT:

Examples of less popular programming languages, with all due respect to their powerful features: Haskell, F#, Ada, etc.

Children:

- [SDOC-SRS-8] *Python language*

7.9.8 Long-term maintainability of a tool

UID:	SDOC-SSS-90
STATUS:	Active

The Requirements Tool shall be designed for long-term maintenance.

NOTE: Long-term maintenance aspects to consider:

- Careful selection of the technologies used, e.g., avoid building on too many unrelated technologies at the same time.
- Take into account the existing experience of the development team. Consider the availability of qualified developers in the future.
- Take into account maintainability by the development team as well as the users, e.g., IT/DevOps department.

Children:

- `[SDOC-SRS-73]` *Standalone ReqIF layer*
- `[SDOC-SRS-14]` *No heavy UI frameworks*
- `[SDOC-SRS-15]` *No use of NPM*
- `[SDOC-SRS-16]` *No use of JavaScript replacement languages (e.g., Typescript)*
- `[SDOC-SRS-42]` *Compliance with Python community practices (PEP8 etc)*

7.10 Text-based requirements language (optional)

Note: Not all requirements tools must be text-based. But when they are, the following requirements apply.

7.10.1 Text files for storing documentation and requirements

UID:	SDOC-SSS-88
STATUS:	Active

The Requirements Tool shall allow storage of documentation and requirements content using text files.

Children:

- `[SDOC-SRS-18]` *Data model*
- `[SDOC-SRS-20]` *SDoc markup language*

7.10.2 Strict text language syntax

UID:	SDOC-SSS-55
STATUS:	Active

The Requirements Tool shall provide a strict syntax for its text language.

Children:

- [SDOC-SRS-19] *Fixed grammar*
- [SDOC-SRS-23] *No indentation*
- [SDOC-SRS-25] *Type-safe fields*

7.10.3 Machine-readable format

UID:	SDOC-SSS-54
STATUS:	Active

The Requirement Tool's text language shall be machine-readable.

Parents:

- [ZEP-2] *Clear separation of requirements (machine-readable)*

Children:

- [SDOC-SRS-19] *Fixed grammar*

7.10.4 Requirements data from multiple repositories

UID:	SDOC-SSS-34
STATUS:	Active

The Requirement Tool shall allow reading requirements files from multiple folders or repositories.

NOTE: The folders/repositories can be arbitrarily nested.

Children:

- [SDOC-SRS-115] *Finding documents recursively*

7.11 Requirements and source code

7.11.1 Traceability between requirements and source code

UID:	SDOC-SSS-72
STATUS:	Active

The Requirements Tool shall support bi-directional tracing between requirements content and implementation source code with only minimal changes needed in the source code.

NOTE: The Requirements Tool does not necessarily have to implement the complete tracing process. It may delegate parts of the traceability task to other tools, e.g., Doxygen, Lobster, etc.

RATIONALE:

This requirement connects the worlds of requirements and source code which ensures that the traceability between requirements-implementation and requirements-tests can be achieved in an explicit way. Without a direct support of tracing requirements to source code by a requirements tool, the users have to find workarounds that are less efficient.

Parents:

- [ZEP-11] *Requirements to source code traceability*
- [ZEP-12] *Non-intrusive links in source code*

Children:

- [SDOC-SRS-33] *Link requirements with source files*
- [SDOC-SRS-34] *Annotate source file*
- [SDOC-SRS-124] *Single-line code marker*
- [SDOC-SRS-35] *Generate source coverage*
- [SDOC-SRS-36] *Generate source file traceability*

7.12 Requirements exchange formats (export/import)

This section captures the requirements related to “Requirements exchange” as outlined in the section *Workflows*.

The Requirements Tool should fundamentally support the exchange of documentation and requirements with other tools. Importing data into this tool and exporting data from it to other tools should be straightforward. The key focus of this section’s requirements is on enabling seamless access to requirements and documentation data.

7.12.1 ReqIF export/import

UID:	SDOC-SSS-58
STATUS:	Active

The Requirements Tool shall support exporting/importing requirements content from/to ReqIF format.

RATIONALE:

ReqIF is a standard for exchanging requirements. There is currently no other standard of a higher maturity.

Parents:

- [ZEP-6] *ReqIF export*

Children:

- [SDOC-SRS-18] *Data model*
- [SDOC-SRS-72] *Export/import from/to ReqIF*

7.12.2 CSV export/import

UID:	SDOC-SSS-59
STATUS:	Active

The Requirements Tool shall support exporting/importing requirements content from/to CSV.

Parents:

- [ZEP-7] *CSV*

Children:

- [SDOC-SRS-129] *Export/import to CSV*

7.12.3 Excel export/import

UID:	SDOC-SSS-60
STATUS:	Active

The Requirements Tool shall support exporting/importing requirements content from/to Excel.

Children:

- [SDOC-SRS-74] *Export to Excel*
- [SDOC-SRS-134] *Selected fields export*

7.13 Collaboration on requirements

7.13.1 Support user accounts

UID:	SDOC-SSS-65
STATUS:	Draft

Children:

- [SDOC-SRS-130] *User accounts*

7.13.2 Send notifications about updated requirements

UID:	SDOC-SSS-66
STATUS:	Draft

Children:

- [SDOC-SRS-131] *Update notifications*

7.14 Development process

7.14.1 Requirements engineering

UID:	SDOC-SSS-76
STATUS:	Active

The Requirements Tool's development process shall include the Requirements Tool's own requirements engineering.

RATIONALE:

A requirements tool is not a trivial project. A clear set of requirements for the developed tool helps to structure the development and communicate the functions of the tool to the developers and the users of the tool.

Children:

- [SDOC-SRS-128] *Requirements-based development*

7.14.2 Self-hosted requirements

UID:	SDOC-SSS-50
STATUS:	Active

The Requirements Tool's requirements shall be developed and stored using the Requirements Tool itself.

RATIONALE:

While not strictly necessary, developing the requirements for the tool using the tool itself aids developers in test-driving its functionality during the requirement development phase. Moreover, having the tool host its own requirements provides a tangible and dynamic illustration of how the tool can be employed for crafting requirements documentation.

Parents:

- [ZEP-15] *Tool Qualifiability*

Children:

- [SDOC-SRS-91] *Self-hosted requirements*

7.14.3 Test coverage

UID:	SDOC-SSS-77
STATUS:	Active

The Requirements Tool's development process shall ensure:

- A testability of the tool.
- The highest possible coverage of the tool's code by test.
- Usage of modern testing methods to ensure adequate coverage of the tool's functions (e.g., command-line interface, web interface, smallest units of code, etc.).

RATIONALE:

The presence of tests, the adequate selection of test methods and a high test coverage are preconditions for a high quality of the requirements tool.

Children:

- [SDOC-SRS-44] *Unit testing*
- [SDOC-SRS-45] *CLI interface black-box integration testing*
- [SDOC-SRS-46] *Web end-to-end testing*
- [SDOC-SRS-47] *At least one integration or end-to-end test*

7.14.4 Tool qualification

UID:	SDOC-SSS-78
STATUS:	Active

The Requirements Tool's development process shall ensure that the tool can be qualified for the use in critical product developments as required by the rigorous technical standards (e.g., EN IEC 61508).

RATIONALE:

Many project developments require a qualification of the tools used during the development. A requirements tool is one of the critical tools that affect the project development. If a requirement tool is developed to the higher standards of quality, it simplifies the argument of bringing the tool forward and using it in a particular project.

Children:

- [SDOC-SRS-6] *Warnings are errors*
- [SDOC-SRS-133] *Priority handling of critical issues in StrictDoc*
- [SDOC-SRS-128] *Requirements-based development*
- [SDOC-SRS-91] *Self-hosted requirements*
- [SDOC-SRS-40] *Use of asserts*
- [SDOC-SRS-41] *Use of type annotations in Python code*
- [SDOC-SRS-43] *Static type checking*
- [SDOC-SRS-44] *Unit testing*
- [SDOC-SRS-45] *CLI interface black-box integration testing*
- [SDOC-SRS-46] *Web end-to-end testing*
- [SDOC-SRS-47] *At least one integration or end-to-end test*

7.15 Licensing and distribution

This section outlines the requirements for the “free and open source” aspect of the Requirements Tool.

7.15.1 Open source

UID:	SDOC-SSS-38
STATUS:	Active

The Requirements Tool’s source code shall be publicly available, e.g., hosted on a code hosting platform such as GitHub or GitLab.

Children:

- [SDOC-SRS-12] *GitHub*

7.15.2 Only open source dependencies

UID:	SDOC-SSS-39
STATUS:	Active

The Requirement Tool’s source code shall be based on open source software components.

Children:

- [SDOC-SRS-89] *Use of open source components*

7.15.3 Free

UID:	SDOC-SSS-40
STATUS:	Active

The Requirements Tool shall be licensed under a permissive license, ensuring no/minimal constraints on the utilization and dissemination of the project.

NOTE: Example of a permissive license: MIT, Apache 2.

RATIONALE:

This requirement captures the essence of an open and free requirements management tool.

Children:

- [SDOC-SRS-118] *StrictDoc license*

8. StrictDoc Requirements Specification (L2)

8.1 SDoc data model

8.1.1 Data model

UID:	SDOC-SRS-18
STATUS:	Active

StrictDoc shall be based on a data model.

RATIONALE:

Designing StrictDoc with a goal of having a consistent data model ensures that the tool:

1) can support a rich set of use cases,
2) model the existing documentation templates used by the industries,
3) interface well with other formats for storing documentation and requirements, e.g., ReqIF and SPDX.

COMMENT:

Verification: data model diagram TBD.

Parents:

- [SDOC-SSS-88] *Text files for storing documentation and requirements*
- [SDOC-SSS-58] *ReqIF export/import*

8.1.2 Requirement model

UID:	SDOC-SRS-26
STATUS:	Active

StrictDoc's data model shall support modeling requirements.

Parents:

- [SDOC-SSS-4] *Requirements CRUD*

8.1.3 Requirement model fields

UID:	SDOC-SRS-100
STATUS:	Active

StrictDoc's "Requirement" model shall support configurable fields system.

Parents:

- [SDOC-SSS-62] *Custom fields*

8.1.4 Requirement model default fields

UID:	SDOC-SRS-132
STATUS:	Active

By default, the Requirement shall support the following fields:

- MID
- UID
- STATUS
- TITLE
- STATEMENT
- RATIONALE
- COMMENT.

RATIONALE:

These fields are the most typical fields found in requirement documents.

Parents:

- [SDOC-SSS-61] *Minimal requirement field set*

8.1.5 Document model

UID:	SDOC-SRS-98
STATUS:	Active

StrictDoc's data model shall support modeling documents.

Parents:

- [SDOC-SSS-64] *Structuring requirements in documents*

8.1.6 Document metadata

UID:	SDOC-SRS-110
STATUS:	Active

StrictDoc's data model shall support a Document metadata model including at least:

- UID
- Document version
- Document classification
- Document authors.

Parents:

- [SDOC-SSS-53] *Document meta information (UID, version, authors, signatures, etc)*
- [SDOC-SSS-75] *Document versioning*

8.1.7 Section model

UID:	SDOC-SRS-99
STATUS:	Active

StrictDoc's data model shall support a concept of a "Section" which nests other Sections, Requirements, Texts.

RATIONALE:

"Section" corresponds to a chapter or a section in a document and helps to organize a document by grouping text nodes, requirements and other sections.

Parents:

- [SDOC-SSS-51] *Documents with nested sections/chapters structure*

8.1.8 Free text

UID:	SDOC-SRS-135
STATUS:	Active

StrictDoc's data model shall support a "Free Text" model, representing non-normative documentation content.

RATIONALE:

Documentation comprises normative components, such as uniquely identifiable elements like requirements or design items, and non-normative components, including introductory text, overview chapters, and other content. The non-normative parts help provide a general understanding for the reader but do not contribute to traceability information. StrictDoc's free text is designed to store this type of non-normative information in SDoc documents.

Parents:

- [SDOC-SSS-3] *Documents (CRUD)*

8.1.9 Composeable document

UID:	SDOC-SRS-109
STATUS:	Active

StrictDoc's data model shall allow composing a Document from other Documents.

RATIONALE:

The logic behind the parent requirement remains fully relevant. Additionally, an alternative approach could involve using a dedicated entity, like "Fragment", to allow a Document to be composed of includable sections or document fragments. Managing composition at the Document level eliminates the need in additional entities like "Fragment", streamlining both the conceptual understanding and the practical implementation of composability.

COMMENT:

The corresponding UI capability for Fragments CRUD is TBD.

Parents:

- `[SDOC-SSS-52]` *Assembling documents from fragments*
- `[DOC178-4]` *Document fragments in separate files*

8.1.10 Requirement relations

UID:	SDOC-SRS-31
STATUS:	Active

The StrictDoc data model shall support connecting requirements using Parent and Child relations.

RATIONALE:

Support of both Parent and Child relations allows to build typical requirements relations such as child-to-parent and less common relations when one document can have parent links to a parent document and child links to a child document (e.g., the so-called "compliance" or "tailoring matrix" documents may use this structure).

Parents:

- `[SDOC-SSS-7]` *Link requirements together*
- `[SDOC-SSS-48]` *Compliance matrices*

8.1.11 Requirement relation roles

UID:	SDOC-SRS-101
STATUS:	Active

Each SDoc relation shall be optionally configurable with a relation role.

NOTE: A relation role is a string value. Typical examples: "refines", "verifies", "implements".

Parents:

- `[SDOC-SSS-8]` *Multiple link roles*

8.2 SDoc text markup

8.2.1 SDoc markup language

UID:	SDOC-SRS-20
STATUS:	Active

StrictDoc shall implement its own text markup language called S-Doc (strict-doc).

RATIONALE:

The most commonly used Markdown format lacks the ability to store requirements metadata. While the RST syntax does allow for customization with directives to implement metadata extensions, its visual appearance contradicts other requirements of StrictDoc, such as the type-safety of the grammar and visual readability. Therefore, a markup language tailored specifically to the needs of the requirements tool provides direct control over the capabilities implemented in both the markup and the user interface.

Parents:

- [SDOC-SSS-88] *Text files for storing documentation and requirements*

8.2.2 Identical SDoc content by import/export roundtrip

UID:	SDOC-SRS-136
STATUS:	Active

StrictDoc shall ensure that identical SDoc content is produced every time StrictDoc reads an SDoc file and then writes it to another SDoc file.

RATIONALE:

A consistent import/export roundtrip implementation and testing reduces the risk of the SDoc bi-directional import/export corruption.

Parents:

- [SDOC-SSS-94] *Data integrity of documentation/requirements*

8.2.3 SDoc and Git storage

UID:	SDOC-SRS-127
STATUS:	Active

StrictDoc shall assume and implement capabilities for storage of SDoc files using Git version control system.

Parents:

- [SDOC-SSS-87] *Programmatic access to requirements data*
- [SDOC-SSS-33] *Version control (Git)*
- [SDOC-SSS-84] *Requirements database*
- [SDOC-SSS-94] *Data integrity of documentation/requirements*

8.2.4 SDoc file extension

UID:	SDOC-SRS-104
STATUS:	Active

The SDoc markup content shall be stored in files with .sdoc extension.

RATIONALE:

Given that the name of the model is S-Doc (strict-doc), it is reasonable to make the document files have the `.sdoc` extension. This helps to identify the document files.

Parents:

- `[SDOC-SSS-80]` *Easy user experience*

8.2.5 One document per one SDoc file

UID:	SDOC-SRS-105
STATUS:	Active

StrictDoc's SDoc file shall represent content of a single document.

COMMENT:

A "Document" corresponds to a "Document" of the SDoc data model.

Parents:

- `[SDOC-SSS-64]` *Structuring requirements in documents*
- `[DO178-1]` *Document concept*

8.2.6 Fixed grammar

UID:	SDOC-SRS-19
STATUS:	Active

StrictDoc's markup language shall be based on a well-defined grammar.

Parents:

- `[DO178-2]` *Strict specified grammar*
- `[SDOC-SSS-55]` *Strict text language syntax*
- `[SDOC-SSS-54]` *Machine-readable format*
- `[SDOC-SSS-94]` *Data integrity of documentation/requirements*

8.2.7 Default grammar fields

UID:	SDOC-SRS-93
STATUS:	Active

The StrictDoc grammar shall have at least the following fields activated by default:

- UID
- STATUS
- LINKS (references to other requirements)
- TITLE
- STATEMENT
- RATIONALE
- COMMENT.

Parents:

- [SDOC-SSS-61] *Minimal requirement field set*

8.2.8 Custom grammar / fields

UID:	SDOC-SRS-21
STATUS:	Active

The SDoc markup shall support custom grammars.

RATIONALE:

A custom grammar allows a user to define their own configuration of requirement fields.

Parents:

- [SDOC-SSS-62] *Custom fields*

8.2.9 Importable grammars

UID:	SDOC-SRS-122
STATUS:	Active

StrictDoc shall support an inclusion of a grammar stored in a separate file.

RATIONALE:

A single grammar defined for several documents helps to standardize the structure of all documents in a documentation tree and removes the effort needed to create identical grammars all the time.

Parents:

- [D0178-9] *Project-level grammar*
- [SDOC-SSS-52] *Assembling documents from fragments*

8.2.10 UID identifier format

UID:	SDOC-SRS-22
STATUS:	Active

The SDoc markup shall only accept UID identifiers that consist of alphanumeric characters separated by a limited set of (“_”, “-”, “.”) characters (TBD).

RATIONALE:

A standardized UID format supports easier unique identification of requirements. It is easier to visually identify UIDs that look similar and common to a given industry.

COMMENT:

This requirement may need a revision to accommodate for more UID formats.

Parents:

- `[SDOC-SSS-89]` *Unique identification of requirements*

8.2.11 Support RST markup

UID:	SDOC-SRS-24
STATUS:	Active

StrictDoc shall support the RST markup.

Parents:

- `[SDOC-SSS-63]` *Text formatting capabilities*

8.2.12 MathJAX

UID:	SDOC-SRS-27
STATUS:	Active

StrictDoc’s markup shall enable support integration with MathJax.

Parents:

- `[SDOC-SSS-63]` *Text formatting capabilities*

8.2.13 No indentation

UID:	SDOC-SRS-23
STATUS:	Active

SDoc text markup blocks shall all start from column 1, i.e., the nesting of the blocks is not allowed.

RATIONALE:

Nesting large text blocks of free text and requirements compromises readability.

Parents:

- [SDOC-SSS-55] *Strict text language syntax*

8.2.14 Type-safe fields

UID:	SDOC-SRS-25
STATUS:	Active

SDoc markup shall provide “type safety” for all fields.

NOTE: “Type safety” means that each field has a type and a corresponding set of validation checks.

Parents:

- [SDOC-SSS-55] *Strict text language syntax*
- [SDOC-SSS-94] *Data integrity of documentation/requirements*

8.3 Graph database

8.3.1 Traceability index

UID:	SDOC-SRS-28
STATUS:	Active

StrictDoc shall maintain a complete Traceability Index of all documentation- and requirements-related information available in a project tree.

Parents:

- [SDOC-SSS-7] *Link requirements together*

8.3.2 Uniqueness UID in tree

UID:	SDOC-SRS-29
STATUS:	Active

For each requirement node, the Traceability Index shall ensure its uniqueness throughout the node’s lifecycle.

RATIONALE:

The requirement ensures that the Traceability Index takes of care of validating the uniqueness of all nodes in a document/requirements graph.

Parents:

- [SDOC-SSS-89] *Unique identification of requirements*
- [SDOC-SSS-94] *Data integrity of documentation/requirements*

8.3.3 Detect links cycles

UID:	SDOC-SRS-30
STATUS:	Active

The Traceability Index shall detect cycles between requirements.

Parents:

- [SDOC-SSS-47] *Requirements database consistency checks*
- [SDOC-SSS-94] *Data integrity of documentation/requirements*

8.3.4 Link document nodes

UID:	SDOC-SRS-32
STATUS:	Active

The Traceability Index shall recognize and maintain the relations between all documents of a project tree.

RATIONALE:

The relations between all documents are a summary of all relations between these documents' requirements. This information is useful for:

1) Structural analysis of a requirements/documents graph.
2) Incremental regeneration of only those documents whose content was modified.

Parents:

- [SDOC-SSS-47] *Requirements database consistency checks*
- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.3.5 Automatic resolution of reverse relations

UID:	SDOC-SRS-102
STATUS:	Active

The StrictDoc's graph database shall maintain the requirement relations and their reverse relations as follows:

- For a Parent relation, the database shall calculate the reverse Child relation.
- For a Child relation, the database shall calculate the reverse Parent relation.

RATIONALE:

The calculation of the reverse relations allows the user interface code to get and display both requirement's parent and child relations.

COMMENT:

Example: If a child requirement REQ-002 has a parent requirement REQ-001, the graph database first reads the link `REQ-002 -Parent> REQ-001`, then it creates a corresponding `REQ-001 -Child> REQ-002` on the go. Both relations can be queried as follows, in pseudocode:

```
get_parent_requirements(REQ-002) == [REQ-001]
get_children_requirements(REQ-001) == [REQ-002]
```

Parents:

- `[SDOC-SSS-71]` *Reverse parent links*
- `[SDOC-SSS-48]` *Compliance matrices*

8.4 Documentation tree

8.4.1 Finding documents recursively

UID:	SDOC-SRS-115
STATUS:	Active

StrictDoc shall discover SDoc documents recursively based on a specified input path.

RATIONALE:

Recursive search allows working with documents located in multiple folders, potentially spanning over several Git repositories.

Parents:

- `[SDOC-SSS-34]` *Requirements data from multiple repositories*
- `[DO178-3]` *Multiple git repositories document assembly*

8.5 Web/HTML frontend

8.5.1 General export requirements

8.5.1.1 Export to static HTML website

UID:	SDOC-SRS-49
STATUS:	Active

StrictDoc shall support generating requirements documentation into static HTML.

Parents:

- `[SDOC-SSS-30]` *Static HTML export*

8.5.1.2 Web interface

UID:	SDOC-SRS-50
STATUS:	Active

StrictDoc shall provide a web interface.

Parents:

- [SDOC-SSS-31] *Graphical user interface (GUI)*
- [DO178-6] *Graphical user interface (GUI)*
- [SDOC-SSS-79] *General usability*
- [SDOC-SSS-80] *Easy user experience*

8.5.1.3 Export to printable HTML pages (HTML2PDF)

UID:	SDOC-SRS-51
STATUS:	Active

StrictDoc shall provide export to printable HTML pages.

Parents:

- [DO178-5] *PDF and HTML publishing*

8.5.1.4 Preserve generated file names

UID:	SDOC-SRS-48
STATUS:	Active

For all export operations, StrictDoc shall maintain the original filenames of the documents when producing output files.

RATIONALE:

Name preservation helps to visually identify which input file an output file corresponds to.

Parents:

- [SDOC-SSS-80] *Easy user experience*

8.5.2 Screen: Project tree

8.5.2.1 View project tree

UID:	SDOC-SRS-53
STATUS:	Active

StrictDoc's "Project tree" screen shall provide browsing of a documentation project tree.

RATIONALE:

This screen is the main tool for visualizing the project tree structure.

Parents:

- [SDOC-SSS-91] *Browsing documentation tree*

8.5.2.2 Create document

UID:	SDOC-SRS-107
STATUS:	Active

StrictDoc's Project Tree screen shall allow creating documents.

Parents:

- [SDOC-SSS-3] *Documents (CRUD)*

8.5.2.3 Delete document

UID:	SDOC-SRS-108
STATUS:	Active

StrictDoc's Project Tree screen shall allow deleting documents.

Parents:

- [SDOC-SSS-3] *Documents (CRUD)*

8.5.3 Screen: Document (DOC)

8.5.3.1 Read document

UID:	SDOC-SRS-54
STATUS:	Active

StrictDoc's Document screen shall allow reading documents.

Parents:

- [SDOC-SSS-3] *Documents (CRUD)*

8.5.3.2 Update document

UID:	SDOC-SRS-106
STATUS:	Active

StrictDoc's Document screen shall allow updating documents.

Parents:

- [SDOC-SSS-3] *Documents (CRUD)*

8.5.3.3 Edit requirement nodes

UID:	SDOC-SRS-55
STATUS:	Active

StrictDoc's Document screen shall allow editing requirements.

Parents:

- [SDOC-SSS-4] *Requirements CRUD*

8.5.3.4 Move requirement / section nodes within document

UID:	SDOC-SRS-92
STATUS:	Active

StrictDoc's Document screen shall provide a capability to move the nodes within a document.

RATIONALE:

Moving the nodes within a document is a convenience feature that speeds up the requirements editing process significantly.

Parents:

- [SDOC-SSS-5] *Move requirement nodes within document*

8.5.3.5 Edit Document grammar

UID:	SDOC-SRS-56
STATUS:	Active

StrictDoc's screen shall allow editing a document's grammar.

RATIONALE:

Editing document grammar allows a user to customize the requirements fields.

Parents:

- [SDOC-SSS-62] *Custom fields*

8.5.3.6 Edit Document options

UID:	SDOC-SRS-57
STATUS:	Active

StrictDoc's Document screen shall provide controls for configuring the document-specific options.

Parents:

- [SDOC-SSS-93] *Document-level configuration*

8.5.3.7 Auto-generate requirements UIDs

UID:	SDOC-SRS-96
STATUS:	Progress

StrictDoc's Document screen shall provide controls for automatic generation of requirements UIDs.

Parents:

- [SDOC-SSS-6] *Auto-provision of Requirement UIDs*
- [SDOC-SSS-80] *Easy user experience*

8.5.3.8 Buttons to copy text to buffer

UID:	SDOC-SRS-59
STATUS:	Active

StrictDoc shall provide a "copy text to buffer" button for all requirement's text fields.

Parents:

- [SDOC-SSS-80] *Easy user experience*

8.5.4 Screen: Table (TBL)

8.5.4.1 View TBL screen

UID:	SDOC-SRS-62
STATUS:	Active

StrictDoc's Table screen shall allow reading documents in a table-like manner.

Parents:

- [SDOC-SSS-73] *Excel-like viewing and editing of requirements*

8.5.5 Screen: Traceability (TR)

8.5.5.1 View TR screen

UID:	SDOC-SRS-65
STATUS:	Active

StrictDoc shall provide a single document-level traceability screen.

NOTE: This screen helps to read a document like a normal document while the traceability to this document's parent and child elements is visible at the same time.

Parents:

- [SDOC-SSS-28] *Traceability matrices*

8.5.6 Screen: Deep traceability (DTR)

8.5.6.1 View DTR screen

UID:	SDOC-SRS-66
STATUS:	Active

StrictDoc shall provide a deep traceability screen.

Parents:

- [DOC178-12] *Uncovered requirement report*

8.5.7 Screen: Project statistics

8.5.7.1 Display project statistics

UID:	SDOC-SRS-97
STATUS:	Active

StrictDoc shall provide a Project Statistics screen that displays the following project information:

- Project title
- Date of generation
- Git revision
- Total documents
- Total requirements
- Requirements status breakdown
- Total number of TBD/TBC found in documents.

RATIONALE:

TBD

Parents:

- [SDOC-SSS-49] *Progress report*
- [DOC178-12] *Uncovered requirement report*
- [SDOC-SSS-29] *Requirements coverage*

8.5.8 Screen: Traceability matrix

8.5.8.1 Traceability matrix

UID:	SDOC-SRS-112
STATUS:	Active

StrictDoc shall provide a traceability matrix screen.

Parents:

- [SDOC-SSS-28] *Traceability matrices*
- [DO178-10] *Traceability matrices*
- [DO178-12] *Uncovered requirement report*

8.5.9 Screen: Project tree diff

8.5.9.1 Project tree diff

UID:	SDOC-SRS-111
STATUS:	Active

StrictDoc shall provide a project tree diff screen.

Parents:

- [SDOC-SSS-75] *Document versioning*
- [SDOC-SSS-74] *Change management*
- [DO178-15] *Diff between document trees*

8.6 Requirements-to-source traceability

8.6.1 Link requirements with source files

UID:	SDOC-SRS-33
STATUS:	Active

StrictDoc shall support bi-directional linking requirements with source files.

Parents:

- [SDOC-SSS-72] *Traceability between requirements and source code*

8.6.2 Annotate source file

UID:	SDOC-SRS-34
STATUS:	Active

StrictDoc shall support a dedicated markup language for annotating source code with links referencing the requirements.

Parents:

- [SDOC-SSS-72] *Traceability between requirements and source code*

8.6.3 Single-line code marker

UID:	SDOC-SRS-124
STATUS:	Active

StrictDoc's source file marker syntax shall support single-line markers.

NOTE: A single-line marker points to a single line in a source file.

RATIONALE:

The advantage of a single-line marker compared to a range marker is that a single-line marker is not intrusive and does not clutter source code. Such a single-marker can be kept in a comment to a function (e.g., Doxygen), not in the function body.

Parents:

- [SDOC-SSS-72] *Traceability between requirements and source code*

8.6.4 Generate source coverage

UID:	SDOC-SRS-35
STATUS:	Active

StrictDoc shall generate project source code coverage information.

NOTE: Source code information can be visualized using both web or CLI interfaces.

Parents:

- [SDOC-SSS-72] *Traceability between requirements and source code*
- [DO178-13] *Source file coverage*

8.6.5 Generate source file traceability

UID:	SDOC-SRS-36
STATUS:	Active

StrictDoc shall generate single file traceability information.

RATIONALE:

With this capability in place, it is possible to focus on a single implementation file's links to requirements which helps in the code reviews and inspections.

Parents:

- [SDOC-SSS-72] *Traceability between requirements and source code*

8.7 Export/import formats

8.7.1 RST

8.7.1.1 Export to RST

UID:	SDOC-SRS-70
STATUS:	Active

StrictDoc shall allow exporting SDoc content to the RST format.

RATIONALE:

Exporting SDoc content to RST enables:

1) Generating RST to Sphinx HTML documentation.
2) Generating RST to PDF using Sphinx/LaTeX.

Parents:

- [D0178-5] *PDF and HTML publishing*
- [D0178-16] *Interoperability with Sphinx*

8.7.1.2 Docutils

UID:	SDOC-SRS-71
STATUS:	Active

StrictDoc shall generate RST markup to HTML using Docutils.

RATIONALE:

Docutils is the most mature RST-to-HTML converter.

COMMENT:

TBD: Move this to design decisions.

Parents:

- [D0178-5] *PDF and HTML publishing*
- [D0178-16] *Interoperability with Sphinx*

8.7.2 ReqIF

8.7.2.1 Export/import from/to ReqIF

UID:	SDOC-SRS-72
STATUS:	Progress

StrictDoc shall support exporting/importing requirements content from/to ReqIF format.

Parents:

- [SDOC-SSS-58] *ReqIF export/import*

8.7.2.2 Standalone ReqIF layer

UID:	SDOC-SRS-73
STATUS:	Active

StrictDoc shall maintain the core ReqIF implementation as a separate software component.

RATIONALE:

ReqIF is a well-defined standard which exists independently of StrictDoc's development. It is reasonable to maintain the ReqIF codebase as a separate software component to allow independent development and easier maintainability.

Parents:

- [SDOC-SSS-90] *Long-term maintainability of a tool*

8.7.3 Excel and CSV

8.7.3.1 Export to Excel

UID:	SDOC-SRS-74
STATUS:	Active

StrictDoc shall allow exporting SDoc documents to Excel, one Excel sheet per document.

Parents:

- [SDOC-SSS-60] *Excel export/import*

8.7.3.2 Selected fields export

UID:	SDOC-SRS-134
STATUS:	Active

StrictDoc Excel export shall allow exporting SDoc documents to Excel with only selected fields.

Parents:

- [SDOC-SSS-60] *Excel export/import*

8.7.4 Graphviz/Dot export

8.7.4.1 Export to Graphviz/Dot

UID:	SDOC-SRS-90
STATUS:	Active

StrictDoc shall support exporting requirements information to PDF format using Graphviz.

RATIONALE:

Graphviz is one of the most capable tools for visualizing graph information, which makes it a perfect tool for visualizing requirements graphs create in StrictDoc.

Parents:

- [SDOC-SSS-56] *1000-feet view*

8.8 Command-line interface

8.8.1 General CLI requirements

8.8.1.1 Command-line interface

UID:	SDOC-SRS-103
STATUS:	Active

StrictDoc shall provide a command-line interface.

Parents:

- [SDOC-SSS-32] *Command-line interface*

8.8.2 Command: Manage

8.8.2.1 Command: Auto UID

8.8.2.1.1 Auto-generate requirements UIDs

UID:	SDOC-SRS-85
STATUS:	Active

StrictDoc shall allow automatic generation of requirements UIDs.

Parents:

- [SDOC-SSS-6] *Auto-provision of Requirement UIDs*

8.9 Python API

8.9.1 StrictDoc Python API

UID:	SDOC-SRS-125
STATUS:	Active

StrictDoc shall provide a Python API for its core functions:

- Reading SDoc files
- Creating traceability graph
- Generating HTML exports
- Converting SDoc to other formats.

Parents:

- `[SDOC-SSS-79]` *General usability*
- `[SDOC-SSS-86]` *Programming access via API (SDK)*
- `[SDOC-SSS-87]` *Programmatic access to requirements data*

8.10 Web server

8.10.1 Web server

UID:	SDOC-SRS-126
STATUS:	Active

StrictDoc shall provide a web server.

RATIONALE:

A web server is a precondition for StrictDoc's web interface. A web server can be available to a single user on their local machine or it can be deployed to a network and be made accessible by several computers.

Parents:

- `[SDOC-SSS-83]` *Server-based deployments (IT-friendly setup)*

8.11 User experience

8.11.1 Strict mode by default

8.11.1.1 Warnings are errors

UID:	SDOC-SRS-6
STATUS:	Active

StrictDoc's default mode of operation shall treat all warnings as errors.

Parents:

- [SDOC-SSS-78] *Tool qualification*

8.12 Configurability

8.12.1 strictdoc.toml file

UID:	SDOC-SRS-37
STATUS:	Active

StrictDoc shall support a configuration of project-level options through a TOML file named `strictdoc.toml`.

Parents:

- [SDOC-SSS-92] *Project-level configuration*

8.12.2 Feature toggles

UID:	SDOC-SRS-39
STATUS:	Active

StrictDoc shall allow a user to select a subset of StrictDoc's available features by listing them in the `strictdoc.toml` file.

Parents:

- [SDOC-SSS-92] *Project-level configuration*

8.12.3 'Host' parameter

UID:	SDOC-SRS-119
STATUS:	Active

StrictDoc shall support configuring a host/port on which the StrictDoc web server is run.

Parents:

- [DOI78-8] *Configuration: 'Host' parameter*

8.13 Performance

8.13.1 Process-based parallelization

UID:	SDOC-SRS-1
STATUS:	Active

StrictDoc shall support process-based parallelization for time-critical tasks.

RATIONALE:

Process-based parallelization can provide a good speed-up when several large documents have to be generated.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.13.2 Caching of parsed SDoc documents

UID:	SDOC-SRS-95
STATUS:	Active

StrictDoc shall implement caching of parsed SDoc documents.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.13.3 Incremental generation of documents

UID:	SDOC-SRS-2
STATUS:	Active

StrictDoc shall support incremental generation of documents.

NOTE: “Incremental” means that only the modified documents are regenerated when StrictDoc is run repeatedly against the same project tree.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.13.4 Caching of RST fragments

UID:	SDOC-SRS-3
STATUS:	Active

StrictDoc shall cache the RST fragments rendered to HTML.

RATIONALE:

Conversion of RST markup to HTML is a time consuming process. Caching the rendered HTML of each fragment helps to save time when rendering the HTML content.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.13.5 On-demand loading of HTML pages

UID:	SDOC-SRS-4
STATUS:	Active

StrictDoc's web interface shall generate the HTML content only when it is directly requested by a user.

RATIONALE:

Generating a whole documentation tree for a user project can be time consuming. The on-demand loading ensures the "do less work" approach when it comes to rendering the HTML pages.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.13.6 Precompiled Jinja templates

UID:	SDOC-SRS-5
STATUS:	Active

StrictDoc shall support a precompilation of HTML templates.

RATIONALE:

The StrictDoc-exported HTML content visible to a user is assembled from numerous small HTML fragments. Precompiling the HTML templates from which the content gets rendered improves the performance of the HTML rendering.

Parents:

- [SDOC-SSS-13] *Support large requirements sets*
- [SDOC-SSS-14] *Support large project trees*

8.14 Development process requirements

8.14.1 General process

8.14.1.1 Priority handling of critical issues in StrictDoc

UID:	SDOC-SRS-133
STATUS:	Active

All critical issues reported in relation to StrictDoc shall be addressed with utmost priority.

RATIONALE:

Prioritizing major issues ensures StrictDoc remains stable and reliable, preventing serious problems that could compromise its performance and integrity.

Parents:

- [SDOC-SSS-78] *Tool qualification*

8.14.2 Requirements engineering

8.14.2.1 Requirements-based development

UID:	SDOC-SRS-128
STATUS:	Active

StrictDoc’s development shall be requirements-based.

Parents:

- [SDOC-SSS-78] *Tool qualification*
- [SDOC-SSS-76] *Requirements engineering*

8.14.2.2 Self-hosted requirements

UID:	SDOC-SRS-91
STATUS:	Active

StrictDoc’s requirements shall be written using StrictDoc.

Parents:

- [SDOC-SSS-50] *Self-hosted requirements*
- [SDOC-SSS-78] *Tool qualification*

8.14.3 Implementation requirements

8.14.3.1 Programming languages

8.14.3.1.1 Python language

UID:	SDOC-SRS-8
STATUS:	Active

StrictDoc shall be written in Python.

RATIONALE:

Python has an excellent package ecosystem. It is a widely used language. It is most often the next language for C/C++ programming community when it comes to the tools development and scripting tasks.

Parents:

- [SDOC-SSS-69] *Conservative languages for implementation*

8.14.3.2 Cross-platform availability

8.14.3.2.1 Linux

UID:	SDOC-SRS-9
STATUS:	Active

StrictDoc shall support the Linux operating systems.

Parents:

- [SDOC-SSS-67] *Support major operating systems*

8.14.3.2.2 macOS

UID:	SDOC-SRS-10
STATUS:	Active

StrictDoc shall support the macOS operating system.

Parents:

- [SDOC-SSS-67] *Support major operating systems*

8.14.3.2.3 Windows

UID:	SDOC-SRS-11
STATUS:	Active

StrictDoc shall support the Windows operating system.

Parents:

- [SDOC-SSS-67] *Support major operating systems*

8.14.4 Implementation constraints

8.14.4.1 Use of open source components

UID:	SDOC-SRS-89
STATUS:	Active

StrictDoc shall be built using only open source software components.

RATIONALE:

No commercial/proprietary dependency chain ensures that StrictDoc remain free and open for everyone.

Parents:

- [DO178-7] *No use of proprietary technology*
- [SDOC-SSS-39] *Only open source dependencies*

8.14.4.2 No heavy UI frameworks

UID:	SDOC-SRS-14
STATUS:	Active

StrictDoc shall avoid using large and demanding UI frameworks.

NOTE: An example of frameworks that require a very specific architecture: React JS, AngularJS.

Parents:

- [SDOC-SSS-90] *Long-term maintainability of a tool*

8.14.4.3 No use of NPM

UID:	SDOC-SRS-15
STATUS:	Active

StrictDoc shall avoid extending its infrastructure with anything based on NPM-ecosystem.

RATIONALE:

StrictDoc already deals with the Python/Pip/Pypi ecosystem. The amount of necessary maintenance is already quite high. NPM is known for splitting its projects into very small parts, which increases the complexity of maintaining all dependencies.

Parents:

- [SDOC-SSS-90] *Long-term maintainability of a tool*

8.14.4.4 No use of JavaScript replacement languages (e.g., Typescript)

UID:	SDOC-SRS-16
STATUS:	Active

StrictDoc shall avoid using JavaScript-based programming languages.

RATIONALE:

The development team does not have specific experience with any of the JS alternatives. Staying with a general subset of JavaScript is a safer choice.

Parents:

- [SDOC-SSS-90] *Long-term maintainability of a tool*

8.14.4.5 Monolithic application with no microservices

UID:	SDOC-SRS-87
STATUS:	Active

StrictDoc shall avoid using microservices and microservice-based architectures.

RATIONALE:

The project is too small to scale to a multi-service architecture.

COMMENT:

This requirement could be re-considered only if a significant technical pressure would require the use of microservices.

Parents:

- [SDOC-SSS-82] *Individual use (home PC)*

8.14.4.6 No reliance on containerization

UID:	SDOC-SRS-88
STATUS:	Active

StrictDoc shall avoid using containers and containerization technologies.

RATIONALE:

Containers are significant extra layer of complexity. They are hard to debug.

COMMENT:

This constraint does not block a StrictDoc user from wrapping StrictDoc into their containers.

Parents:

- [SDOC-SSS-82] *Individual use (home PC)*

8.14.5 Coding constraints

8.14.5.1 Use of asserts

UID:	SDOC-SRS-40
STATUS:	Active

StrictDoc's development shall ensure a use of assertions throughout the project codebase.

NOTE: At a minimum, the function input parameters must be checked for validity.

Parents:

- [SDOC-SSS-78] *Tool qualification*

8.14.5.2 Use of type annotations in Python code

UID:	SDOC-SRS-41
STATUS:	Active

StrictDoc's development shall ensure a use of type annotations throughout the project's Python codebase.

Parents:

- [SDOC-SSS-78] *Tool qualification*

8.14.6 Linting

8.14.6.1 Compliance with Python community practices (PEP8 etc)

UID:	SDOC-SRS-42
STATUS:	Active

StrictDoc's development shall ensure that the project's codebase is compliant with the Python community's modern practices.

Parents:

- [SDOC-SSS-90] *Long-term maintainability of a tool*

8.14.7 Static analysis

8.14.7.1 Static type checking

UID:	SDOC-SRS-43
STATUS:	Active

StrictDoc's development shall include a continuous type checking of StrictDoc's codebase.

Parents:

- [SDOC-SSS-78] *Tool qualification*

8.14.8 Testing

8.14.8.1 Unit testing

UID:	SDOC-SRS-44
STATUS:	Active

StrictDoc's development shall provide unit testing of its codebase.

Parents:

- [SDOC-SSS-77] *Test coverage*
- [SDOC-SSS-78] *Tool qualification*

8.14.8.2 CLI interface black-box integration testing

UID:	SDOC-SRS-45
STATUS:	Active

StrictDoc's development shall provide complete black-box integration testing of its command-line interface.

Parents:

- [SDOC-SSS-77] *Test coverage*
- [SDOC-SSS-78] *Tool qualification*

8.14.8.3 Web end-to-end testing

UID:	SDOC-SRS-46
STATUS:	Active

StrictDoc's development shall provide complete end-to-end testing of the web interface.

Parents:

- [SDOC-SSS-77] *Test coverage*

- [SDOC-SSS-78] *Tool qualification*

8.14.8.4 At least one integration or end-to-end test

UID:	SDOC-SRS-47
STATUS:	Active

Every update to the StrictDoc codebase shall be complemented with a corresponding provision of at least one test as follows:

- For web interface: at least one end-to-end test.
- For command-line interface: at least one black-box integration test.
- For internal Python functions: at least one unit test.

NOTE: This requirement implies that no modifications to StrictDoc's functionality can be merged unless accompanied by at least one test.

RATIONALE:

This requirement ensures that every new feature or a chance in the codebase is covered/stressed by at least one test, according to the change type.

Parents:

- [SDOC-SSS-77] *Test coverage*
- [SDOC-SSS-78] *Tool qualification*

8.15 Code hosting and distribution

8.15.1 Code hosting

8.15.1.1 GitHub

UID:	SDOC-SRS-12
STATUS:	Active

StrictDoc's source code shall be hosted on GitHub.

Parents:

- [SDOC-SSS-38] *Open source*
- [SDOC-SSS-82] *Individual use (home PC)*

8.15.2 StrictDoc license

UID:	SDOC-SRS-118
STATUS:	Active

All StrictDoc software shall be licensed under the Apache 2 license.

Parents:

- `[SDOC-SSS-40]` *Free*

9. Design Document

This document describes the architecture and the implementation details of StrictDoc. Compared to the User Guide that describes how to use StrictDoc, this Design Document focuses on the “how it works” of StrictDoc.

9.1 Overview

StrictDoc consists of two applications:

1. StrictDoc command-line application (CLI).
2. StrictDoc web application.

Both applications share a significant subset of the backend and frontend logic. The backend logic is written in Python, the frontend logic is written using HTML/CSS, Jinja templates, and a combination of Turbo.js/Stimulus.js frontend libraries.

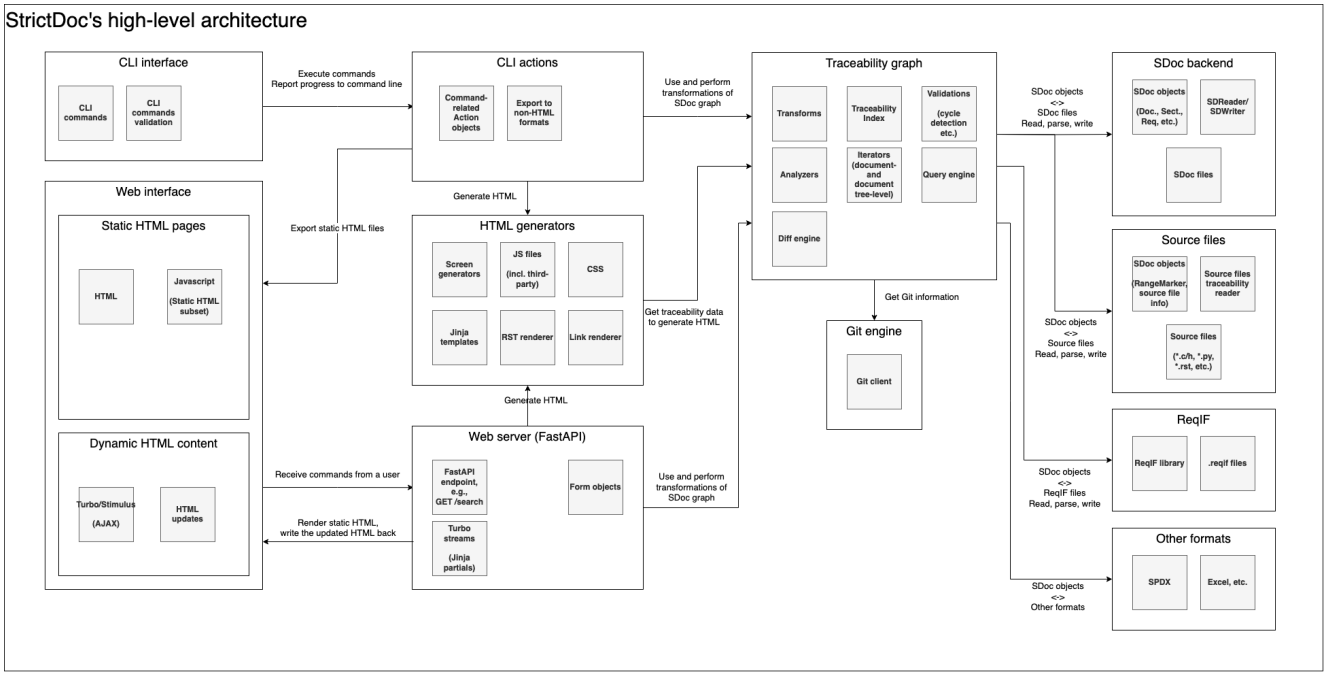
9.2 Building blocks

StrictDoc is based on the following open-source libraries and tools:

Library/tool	Description
TextX	Used for StrictDoc grammar definition and parsing of the sdoc files.
Jinja	Rendering HTML templates.
Sphinx and Docutils	• Support of Restructured Text (reST) format • Generation of RST documents into HTML • Generation of RST documents into PDF using LaTeX • Generating documentation websites using Sphinx.
FastAPI	Server used for StrictDoc’s Web-based user interface.
Turbo and Stimulus	Javascript frameworks used for StrictDoc’s Web-based user interface.
Selenium and SeleniumBase	Used for end-to-end testing of StrictDoc’s Web-based user interface.

9.3 High-level architecture

The following diagram captures the high-level architecture of StrictDoc.



9.4 StrictDoc command-line application

StrictDoc command-line application is at the core of StrictDoc. The command-line interface contains commands for exporting/importing SDoc content from/to other formats and presenting documentation content to a user.

The command-line application can be seen as a Model-View-Controller application:

- A command entered by a user gets recognized by the CLI arguments parser.
- Depending on the type of command, a responsible Action (Controller layer) processes the command (export action, import action, etc.).
- The input of the command is transformed by the action using the backend (Model layer) (SDoc, ReqIF, Excel, etc.).
- The resulting output is written back to HTML or other formats (View layer).

9.5 StrictDoc web application

StrictDoc Web application is based on FastAPI / Uvicorn. The end-to-end usage cycle of the web application is as follows:

- A browser requests documents from a FastAPI server.
- The FastAPI web server parses the SDoc files into memory and converts them into HTML using Jinja templates. The resulting HTML output is given back to the user.
- The Jinja templates are extended with JavaScript logic that allows a user to edit the documents and send the updated content back to the server.
- The server writes the updated content back to the SDoc files stored on a user's file system.

9.5.1 The HTML Over the Wire (Hotwire) architecture

StrictDoc uses the Hotwire architecture.

The JavaScript framework used by StrictDoc is minimized to Turbo.js/Stimulus.js which helps to avoid the complexity of the larger JS frameworks such as React, Vue, Angular, etc. In accordance with the Hotwire approach, most of the StrictDoc's business logic is done on a server, while Turbo and Stimulus provide a thin layer of JS and AJAX to connect the almost static HTML with the server.

The Hotwire approach helps to reduce the differences between the static HTML produced by the StrictDoc command-line application and the StrictDoc web application. In both cases, the core content of StrictDoc is a statically generated website with documents. The web application extends the static HTML content with Turbo/Stimulus to turn it into a dynamic website.

Currently, the web server renders the HTML documents using the same generators that are used by the static HTML export, so the static HTML documentation and the web application interface look identical. The web interface adds the action buttons and other additional UI elements for editing the content.

9.6 Parsing SDoc files

StrictDoc uses textX which is a `meta-language for building Domain-Specific Languages (DSLs) in Python`. The textX itself is based on Arpeggio which is a `Parser interpreter based on PEG grammars written in Python`.

StrictDoc relies on both tools to get:

- A declarative grammar description
- Automatic conversion of the parsed blocks into Python objects
- Fast parsing of SDoc files.

One important implementation detail of Arpeggio that influences StrictDoc user experience is that the parser stops immediately when it encounters an error. For a document that has several issues, the parser highlights only the first error without going any further. When the first error is resolved, the second error will be shown, etc.

10. StrictDoc Backlog

This document outlines the future work items for StrictDoc.

The following items are listed in descending order of priority, with the topmost items either currently in progress or scheduled to be implemented next.

While this backlog overlaps with StrictDoc's GitHub issues tracker by more than 50%, it includes more strategic items compared to the GitHub issues, which are primarily focused on actual implementation work.

10.1 StrictDoc challenges

- Limited development time.
- Not easy to develop certain capabilities and scale to a multi-user environment quickly.

10.1.1 Real-time editing out of scope

UID:	SDOC-SRS-13
STATUS:	Backlog

StrictDoc shall not implement the real-time editing capability to its web interface.

RATIONALE:

The real-time editing feature is hard to achieve with a small part-time involvement from the development team. This requirement can only be reconsidered, if StrictDoc would experience a significant increase in the development power.

10.2 Backlog

10.2.1 Auto-commit to Git repository

UID:	SDOC-BACKLOG-6
STATUS:	Backlog

10.2.2 Auto-generate section UIDs

UID:	SDOC-SRS-86
STATUS:	Backlog

TBD

10.2.3 Screen: Project home

10.2.3.1 View project home page

UID:	SDOC-SRS-52
STATUS:	Backlog

10.2.4 Screen: Traceability navigator

10.2.4.1 Traceability navigator

UID:	SDOC-SRS-113
STATUS:	Backlog

StrictDoc shall provide a traceability navigator screen.

RATIONALE:

Provide an interactive 1000-ft view of a requirements project.

Parents:

- `[SDOC-SSS-56]` *1000-feet view*

10.2.5 Formal modeling

10.2.5.1 Integration with other systems engineering processes

UID:	SDOC-RMC-27
STATUS:	Backlog

The Requirements Tool shall provide capabilities for integration with other systems engineering tools.

10.2.5.2 Integration with Capella

UID:	SDOC-RMC-29
STATUS:	Backlog

The Requirements Tool shall provide integration with Capella MBSE tool.

RATIONALE:

Eclipse Capella is a capable open-source tool for Model-Based Systems Engineering, https://www.eclipse.org/capella/. It should be beneficial for the requirements tool to interface with the Capella engineering community.

COMMENT:

At the very least, the integration can happen through the ReqIF interface that Capella is known to support.

10.2.5.3 Support STPA method

UID:	SDOC-RMC-55
STATUS:	Backlog

The Requirements Tool shall provide support for the STPA method.

10.2.5.4 Formalized statements

UID:	SDOC-RMC-28
STATUS:	Backlog

The Requirements Tool shall provide capabilities for hardening requirements content with formal semantics.

COMMENT:

The directions to explore:

- NASA FRET
- bmw-software-engineering/trlc

10.2.5.5 AI Assistant

UID:	SDOC-RMC-30
STATUS:	Backlog

The Requirements Tool shall provide integration with AI tools (e.g., ChatGPT).

10.2.6 LaTeX export

10.2.6.1 Export to Tex

UID:	SDOC-SRS-76
STATUS:	Backlog

10.2.7 Focused mode: Edit a single section / requirement

UID:	SDOC-BACKLOG-1
STATUS:	Backlog

StrictDoc shall allow focused editing of single sections and requirements on a dedicated screen.

COMMENT:

This is partially implemented with the composable documents feature. An included document can be edited as a standalone document.

10.2.8 Interoperability with Doxygen

UID:	SDOC-BACKLOG-2
STATUS:	Backlog

10.2.9 Fuzzy search (the whole documentation)

UID:	SDOC-BACKLOG-3
STATUS:	Backlog

10.2.10 Derived requirements

UID:	SDOC-BACKLOG-9
STATUS:	Backlog

StrictDoc shall provide first-class support for Derived requirements.

Parents:

- [D0178-18] *Support for Derived requirements*

10.2.11 Support Markdown markup

UID:	SDOC-BACKLOG-4
STATUS:	Backlog

10.2.12 Language Server Protocol (LSP)

UID:	SDOC-BACKLOG-7
STATUS:	Backlog

10.2.13 UML

UID:	SDOC-BACKLOG-8
STATUS:	Backlog

10.2.14 Export/import to CSV

UID:	SDOC-SRS-129
STATUS:	Backlog

StrictDoc shall allow exporting/import SDoc content to/from CSV.

Parents:

- [SDOC-SSS-59] *CSV export/import*

10.2.15 Web API

UID:	SDOC-SRS-114
STATUS:	Backlog

StrictDoc shall provide a web API.

RATIONALE:

A web API allows integration with tools and workflows external to StrictDoc itself.

Parents:

- [SDOC-SSS-68] *Web API interface*
- [SDOC-SSS-79] *General usability*
- [SDOC-SSS-85] *Programming access via API (Web)*

10.2.16 Multi-user workflow

10.2.16.1 Multi-user editing of documents

UID:	SDOC-SRS-123
STATUS:	Backlog

StrictDoc shall support concurrent use and editing of a single StrictDoc web server instance by multiple users.

Parents:

- [DOI78-17] *Multi-user editing of documents*
- [SDOC-SSS-81] *Support projects with a large number of users*

10.2.16.2 User accounts

UID:	SDOC-SRS-130
STATUS:	Backlog

StrictDoc shall support user accounts.

Parents:

- `[SDOC-SSS-65]` *Support user accounts*

10.2.16.3 Update notifications

UID:	SDOC-SRS-131
STATUS:	Backlog

StrictDoc shall support notifying a user (users) about updated requirements.

Parents:

- `[SDOC-SSS-66]` *Send notifications about updated requirements*
- `[SDOC-SSS-74]` *Change management*

10.2.17 Requirement validation according to EARS syntax

UID:	SDOC-SRS-116
STATUS:	Backlog

The SDoc model shall provide validation of requirements according to the EARS syntax.

Parents:

- `[SDOC-SSS-57]` *Requirement syntax validation (e.g. EARS)*

10.2.18 WYSIWYG editing

UID:	SDOC-SRS-121
STATUS:	Backlog

StrictDoc shall provide WYSIWYG kind of editing for all multiline text input fields.

RATIONALE:

WYSIWYG improves the user experience, especially for non-programmer users.

Parents:

- `[DOI78-19]` *WYSIWYG editing*
- `[SDOC-SSS-80]` *Easy user experience*

10.2.19 Tables HTML editor

UID:	SDOC-SRS-61
STATUS:	Backlog

StrictDoc shall provide a solution for editing tables in its web interface.

10.2.20 Move requirement / section nodes between documents

UID:	SDOC-SRS-94
STATUS:	Backlog

StrictDoc's Document screen shall provide a capability to move the nodes between documents.

RATIONALE:

Moving the nodes within a document is a convenience feature that speeds up the requirements editing process significantly.

Parents:

- `[SDOC-SSS-70]` *Move nodes between documents*

10.2.21 Auto-completion for requirements UIDs

UID:	SDOC-SRS-120
STATUS:	Backlog

StrictDoc's Document screen shall provide controls for automatic completion of requirements UIDs.

COMMENT:

The automatic completion can be especially useful when a user has to fill in a parent relation UID.

Parents:

- `[SDOC-SSS-6]` *Auto-provision of Requirement UIDs*
- `[DO178-14]` *Requirement UID autocompletion*
- `[SDOC-SSS-80]` *Easy user experience*

10.2.22 Attach image to requirement

UID:	SDOC-SRS-58
STATUS:	Backlog

10.2.23 Provide contextual help about RST markup

UID:	SDOC-SRS-60
STATUS:	Backlog

10.2.24 TBL: Hide/show columns

UID:	SDOC-SRS-63
STATUS:	Backlog

StrictDoc's Table screen shall allow hiding/showing columns.

10.2.25 TBL: Select/deselect tags

UID:	SDOC-SRS-64
STATUS:	Backlog

StrictDoc's Table screen shall allow filtering content based on the selection/deselection of available tags.

10.2.26 Screen: Impact analysis

10.2.26.1 Impact analysis

UID:	SDOC-SRS-117
STATUS:	Backlog

StrictDoc shall provide the Impact Analysis screen.

NOTE: The Impact Analysis screen helps to get information about the impact that a given change to a requirement has on the other requirements in the project tree.

RATIONALE:

The impact analysis is one of the core functions of a requirements management tool. Analyzing the impact that a requirement has on other requirements and an overall project's technical definition helps to perform effective change management.

Parents:

- `[SDOC-SSS-74]` *Change management*
- `[DO178-11]` *Impact analysis*

10.2.27 ReqXLS

UID:	SDOC-SRS-75
STATUS:	Backlog

10.3 Backlog: Web-based user interface

- Uploading images via Web interface.
- Deleting sections recursively. Correct clean-up of all traceability information.
- Editing remaining document options: Inline/Table, Requirements in TOC, etc.
- **Integration with Git repository.** Make the server commit changes to .sdoc files automatically. To a user, provide visibility to what happens under the hood.
- `LINK` between sections and documents.
- Option to keep all multi-line text fields to 80 symbols width.
- Moving nodes between documents.
- TBL view: Column filters to show/hide columns.
- TBL view: Completely empty columns are hidden by default.
- Contextual help about the RST markup.
- How to edit tables conveniently?
- What to do with web content going out of sync with the server/file system state?
- Issue when adding a child section from a nested section. The child section appears right after the nested section, not after its farthest descendant child.
- ReqIF: Export complete documentation tree or a single document.
- ReqIF: Import complete documentation tree or a single document.

10.4 Backlog: Nice to have

- Configuration file options:
  - CLI command to dump default config file
  - Project prefix?
  - Config options for presenting requirements. - Include/exclude requirements in TOC
- **StrictDoc as a Python library**. Such a use allows a more fine-grained access to the StrictDoc's modules, such as Grammar, Import, Export classes, etc.
- **Data exchange with Capella tool.** The current idea would be to implement this using ReqIF export/import features.
- **Language Server Protocol.** The LSP can enable editing of SDoc files in IDEs like Eclipse, Visual Studio, PyCharm. A smart LSP can enable features like syntax highlighting, autocompletion and easy navigation through requirements. The promising base for the implementation: https://github.com/openlawlibrary/pygls.
- StrictDoc shall support rendering text/code blocks into Markdown syntax.

- **Fuzzy requirements search.** This feature can be implemented in the CLI as well as in the future GUI. A fuzzy requirements search can help to find existing requirements and also identify relevant requirements when creating new requirements.
- Support creation of FMEA/FMECA safety analysis documents.
- Calculation of checksums for requirements. This feature is relatively easy to implement, but the implementation is postponed until the linking between requirements and files is implemented.
- Filtering of requirements by tags.
- Import/export: Excel, CSV, PlantUML, Confluence, Tex, Doorstop.
- **Partial evaluation of Jinja templates.** Many of the template variables could be made to be evaluated once, for example, config object's variables.
- UI version for mobile devices (at least some basic tweaks).

10.5 Backlog: Technical debt

- When a document is added, the whole documentation is rebuilt from the file system from scratch. A more fine-grained re-indexing of documentation tree can be implemented. The current idea is to introduce a layer of pickled cached data: preserve the whole in-memory traceability graph in a cache, and then use the cached data for making decisions about what should be regenerated.
- The "no framework" approach with FastAPI and Turbo/Stimulus allows writing almost zero Javascript, however some proto-framework conventions are still needed. Currently, all code is written in the `main_controller` which combines all responsibilities, such as parsing HTTP request fields, accessing traceability graph, validations, rendering back the updated AJAX templates. A lack of abstraction is better than a poor abstraction, but some solution has to be found.
- Request form object vs Response form object. The workflow of form validations is not optimal.
- For Web development, the responsibilities of the `TraceabilityIndex` class compared to the `ExportAction`, `MarkupRenderer`, `LinkRenderer` classes are unstable. A more ecological composition of classes has to be found. `Traceability` index is rightfully a "god object" because it contains all information about the in-memory documentation graph.

10.6 Open questions

10.6.1 One or many input sdoc trees

StrictDoc supports this for HTML already but not for RST.

When passed `strictdoc export ... /path/to/doctree1, /path/to/doctree2, /path/to/doctree3`, the following is generated:

```
output folder:
- doctree1/
  - contents
- doctree2/
  - contents
- doctree3/
  - contents
```

and all three doctrees' requirements are merged into a single documentation space with cross-linking possible.

The question is if it is worth supporting this case further or StrictDoc should only work with one input folder with a single doc tree.

11. Credits

As an open-source project, StrictDoc is based on the work of many people and organizations:

- StrictDoc receives contributions from other developers.
- StrictDoc is built using other open-source software.
- StrictDoc uses free hosting and Continuous Integration services provided for open-source software.
- StrictDoc uses the commercial versions of JetBrains IDEs for free.

This page gives due credit to everyone who made StrictDoc possible.

11.1 Contributions to StrictDoc

The core team: @stanislaw and @mettta.

The following people and organizations have contributed to StrictDoc. The contributions are listed in the alphabetic order.

- @BenGardiner – Import Excel feature, improvements of HTML and RST export, Document Fragments feature.
- @GGBeer – Generating bibliography with BibTeX (ongoing), improvements of Excel export.
- @lochsh – MathJax support.
- @Relasym – Important fixes of how the documents are re-generated (or not).
- @stumpyfr – Improvements of Excel export.

Companies:

- BUGSENG and @RobertoBagnara have contributed bug reports and feature suggestions.
- Kontrol have sponsored the work related to the early implementation of the ReqIF import/export feature (@alex.d, @cbernt, @Relasym).

Single/smaller contributions can be also seen on the StrictDoc's Insights/Contributors page.

11.2 Open source software

StrictDoc is based on other open source components. Without this support, we would have never reached where we are today.

StrictDoc was heavily inspired by the Doorstop project. Without the strong example of Doorstop, StrictDoc would probably never exist.

StrictDoc uses textX as an underlying parser for the SDoc text markup language.

StrictDoc uses Sphinx and Docutils for generating SDoc documents to RST, HTML and PDF formats.

StrictDoc has a satellite project reqif which is built on top of the lxml parser. The `reqif` library allows StrictDoc to import and export documents from/to ReqIF format.

StrictDoc uses FastAPI and a combination of Turbo.js and Stimulus.js for its Web-based graphical interface. This combination helps StrictDoc to stick with the HTML over the wire approach.

StrictDoc uses Jinja as a templating engine. Jinja is used for both static HTML and RST exports as well as in the Web-based GUI.

StrictDoc uses Pygments to color-code the source files for its “requirements to source files” traceability feature.

StrictDoc uses XlsxWriter and xlrd for its Excel export/import features.

The credits also recursively go to the building blocks of each of the above projects because most of them have their own dependencies.

Refer to the configuration file for an up-to-date summary of StrictDoc’s dependencies.

11.3 Hosting and Continuous Integration

StrictDoc is hosted on GitHub and uses GitHub Actions to run all of its build, test and release tasks. As an open-source project, StrictDoc gets these services from GitHub for free.

StrictDoc’s documentation is hosted on Read the Docs.

11.4 Commercial IDEs by JetBrains

The StrictDoc’s core team uses the commercial versions of PyCharm and WebStorm from JetBrains.

The Licenses for Open Source Development - Community Support from JetBrains are provided to the core team for free, based on the precondition that StrictDoc is developed as completely free software, without any monetization mechanisms.

Before the license for the commercial JetBrains was obtained in 2023, the complete StrictDoc’s Python codebase had been produced using PyCharm, the Community edition.

12. Technical Note: DO-178C requirements tool requirements

This document outlines a set of high-level requirements for StrictDoc, a text-based requirements management system. While StrictDoc already meets many of these requirements, a further discussion is needed to clarify any remaining questions. For the outstanding requirements, we can establish a practical implementation plan within the upcoming 2023-2024 StrictDoc roadmap.

These requirements are recommended by engineers who adhere to the DO-178B and DO-178C standards of the aviation industry. For a visual summary of the DO-178 standard, please refer to this link: https://upload.wikimedia.org/wikipedia/commons/4/4f/DO-178B_Process_Visual_Summary_Rev_A.pdf.

12.1 Already implemented features

12.1.1 Document concept

UID:	DO178-1
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall store requirements in document files.

RATIONALE:

A concept of a "document file with requirements" helps to structure requirements like they are normally structured in the documents.

An alternative implementation of "1 file per 1 requirement" can be very restrictive in some use cases. For example, one needs to open lots of files to edit, if one file can only have one requirement.

Children:

- `[SDOC-SRS-105]` *One document per one SDoc file*

12.1.2 Strict specified grammar

UID:	DO178-2
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall feature a specified document grammar.

RATIONALE:

The grammar helps to standardize a document structure.

COMMENT:

N: StrictDoc is nice.

Children:

- `[SDOC-SRS-19]` *Fixed grammar*

12.1.3 Requirement UID autocompletion

UID:	DO178-14
STATUS:	Active

StrictDoc shall provide autocompletion feature for requirement UID identifiers.

Note: Most immediate use case: adding/editing parent requirements.

COMMENT:

N: When adding parent links, StrictDoc GUI shall present a selection list of UID, with a completion filter, then compute the sha1 of the selected parent req.

COMMENT:

N: Upon req editing, a completion list of already existing reqs (+ “derived” item) would be definitely nice in Webgui ! and would be the ultimate argument to NOT text edit.

Children:

- `[SDOC-SRS-120]` *Auto-completion for requirements UIDs*

12.1.4 Multiple git repositories document assembly

UID:	DO178-3
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support generating requirement trees from multiple Git repositories.

COMMENT:

N: StrictDoc is compliant.

Children:

- `[SDOC-SRS-115]` *Finding documents recursively*

12.1.5 Document fragments in separate files

UID:	DO178-4
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support assembly of documents from multiple files.

COMMENT:

S: StrictDoc supports document fragments. A document fragment corresponds to a section that can be kept in a separate file. A document stored in another file can import the fragment and have it included in the main document.

Children:

- `[SDOC-SRS-109]` *Composeable document*

12.1.6 PDF and HTML publishing

UID:	DO178-5
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support publication of documents to HTML and PDF formats.

COMMENT:

N: Sphinx is nice for release.

Children:

- `[SDOC-SRS-51]` *Export to printable HTML pages (HTML2PDF)*
- `[SDOC-SRS-70]` *Export to RST*
- `[SDOC-SRS-71]` *Docutils*

12.1.7 Graphical user interface (GUI)

UID:	DO178-6
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support a graphical user interface.

COMMENT:

N: A Web GUI in StrictDoc is nice in daily work, especially for non developer people.

COMMENT:

N: GUI for editing is NTH but it shall scale well to thousands of requirements. And it could also contribute to traceability feature.

Children:

- `[SDOC-SRS-50]` *Web interface*

12.1.8 Configuration: 'Host' parameter

UID:	DO178-8
STATUS:	Active

StrictDoc shall provide an option to configure a host where a server is deployed.

COMMENT:

N: Binding to any local address (localhost) with an option would enable to edit from a smartphone bound to a Raspberry server, for instance.

Children:

- `[SDOC-SRS-119]` *'Host' parameter*

12.1.9 No use of proprietary technology

UID:	DO178-7
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall not use any proprietary tools.

RATIONALE:

Use of proprietary tools complicates the workflows and the interoperability between companies and teams.

COMMENT:

S: StrictDoc is written using Python and supports the ReqIF format out of the box. All StrictDoc's dependencies are open-source software components.

Children:

- `[SDOC-SRS-89]` *Use of open source components*

12.1.10 Source file coverage

UID:	DO178-13
STATUS:	Active

StrictDoc shall support generation of source code coverage information.

COMMENT:

S: Source file coverage is StrictDoc's experimental feature. With a more detailed specification, we can turn it to a more advanced and clear presentation of the needed aspects.

Children:

- `[SDOC-SRS-35]` *Generate source coverage*

12.1.11 Project-level grammar

UID:	DO178-9
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support creation of a project-level grammar.

RATIONALE:

A single grammar defined for a project (same grammar for several documents) helps to standardize the structure of all documents in a documentation tree and reduces the effort needed to create identical grammars all the time.

COMMENT:

S: This feature is easy to implement. The easiest implementation path is to include a config parameter, such as `project_grammar` in the already-existing `strictdoc.toml` file. At startup, StrictDoc

recognizes the parameter and reads the grammar from a separate file. The project grammar becomes a single source of truth for all documents in the project tree but the option to override a grammar for a given document is still preserved.

Children:

- [SDOC-SRS-122] *Importable grammars*

12.2 Needs discussion

12.2.1 WYSIWYG editing

UID:	DO178-19
STATUS:	Active

StrictDoc's GUI shall support a WYSIWYG text editing.

COMMENT:

Simplifies editing of formatted text.

Children:

- [SDOC-SRS-121] *WYSIWYG editing*

12.2.2 Diff between document trees

UID:	DO178-15
STATUS:	Active

StrictDoc shall allow calculating Diff between two document trees.

Note: The primary use case is calculating a diff between two Git revisions.

COMMENT:

N: Highlight a req diff with its previous version (Git).

Children:

- [SDOC-SRS-111] *Project tree diff*

12.2.3 Traceability matrices

UID:	DO178-10
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support generation of forward and backward traceability matrices.

COMMENT:

N: Trace matrix publishing (both ways : is covered by ... and covers ...) published in HTML/PDF.

COMMENT:

S: This feature, especially a very basic initial one, is very easy to implement, and it is already on the nearest roadmap, see https://github.com/strictdoc-project/strictdoc/issues/964#issuecomment-1497900436>. We only need to agree on if we are on the same page about how the produced matrices look like.

Children:

- [SDOC-SRS-112] *Traceability matrix*

12.2.4 Impact analysis

UID:	DO178-11
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support generation of Impact Analysis information.

COMMENT:

N: Impact analysis – upon modification of a requirement: report the recursive list of impacted items.

COMMENT:

S: This feature is doable and a basic variant can be derived from the existing code that generates the Deep Traceability screen. A more advanced one includes a document-to-document Diff between version control revisions, including “tell me what changed between the latest commit and my changes”. Based on this information, a full impact analysis package can be generated. This is less trivial to implement and requires prioritization.

COMMENT:

N: For impact analysis we were thinking about some design which help to satisfy these feature: upon modification of a requirement which owns some parent links, a SHA1 of each parent requirement statement is computed and set in the edited requirement. => this could be captured by the GUI, and there also could exist a CLI command to perform this tagging.

For overall analysis, a CLI command could parse the tree and compute the SHA1 and tel which requirement are to be updated because one of there ancestor were modified. This is almost the same feature called review status in doorstop.

COMMENT:

N: When adding parent links, the GUI could present a selection list of UID, with a completion filter, then compute the SHA1 of the selected parent req. Then highlight uncovered requirement, and requirements impacted by parent change.

Children:

- [SDOC-SRS-117] *Impact analysis*

12.2.5 Uncovered requirement report

UID:	DO178-12
COMPLIANCE:	C
STATUS:	Active

StrictDoc shall support generation of uncovered requirement report.

Note: An uncovered requirement is one that has no children.

COMMENT:

S: This is easy to implement but would be nice to have it specified in terms of how exactly it should look like. The requirements coverage screen was one experimental attempt to visualize and highlight the uncovered requirements but we didn't stabilize the feature in terms of the visual clarity.

Children:

- `[SDOC-SRS-66]` *View DTR screen*
- `[SDOC-SRS-97]` *Display project statistics*
- `[SDOC-SRS-112]` *Traceability matrix*

12.2.6 Interoperability with Sphinx

UID:	DO178-16
COMPLIANCE:	PC
STATUS:	Backlog

StrictDoc shall support interoperability with Sphinx:

1) StrictDoc shall read RST fragments with Sphinx directives without errors.
2) StrictDoc shall render Sphinx plugins natively.

COMMENT:

N: Support various fragments (images, csv, doxygen, uml, math expr…) => Sphinx extensions nice.

COMMENT:

S: It should be possible to achieve the goal 1 by implementing a complete or limited behavior of each Sphinx plugin feature like I already suggested here. For each needed plugin, we can implement a simulative directive using Docutils, and I expected that for many plugins we can achieve a good compatible behavior. The goal 2 needs a special R&D activity where it has to be decided what would be the interface between StrictDoc and Sphinx.

COMMENT:

N: `image.*` is MTH to enable both HTML and pdf. breathe is required for the Software Design Description document which defines software architecture, low level requirements and code component interfaces. But it could be Split in 2 separate documents. LLR in .sdoc and code component interface with sphinx/breathe. So I consider it as NTH.

Children:

- `[SDOC-SRS-70]` *Export to RST*
- `[SDOC-SRS-71]` *Docutils*

12.2.7 Multi-user editing of documents

UID:	DO178-17
COMPLIANCE:	NC
STATUS:	Backlog

StrictDoc shall allow multi-user editing of documents.

COMMENT:

N: .sdoc file lock?

Children:

- [SDOC-SRS-123] *Multi-user editing of documents*

12.2.8 Support for Derived requirements

UID:	DO178-18
STATUS:	Backlog

StrictDoc shall provide first-class support for Derived requirements.

COMMENT:

N: I would mention another important feature related to DO178. The requirement which have not parent are “derived” and shall be assessed by safety.

Two issues when a parent ref is set to `REQUIRED: True` in grammar:

1. I cannot specify derived requirements.
2. Top reqs do not have parents by définition.

I worked around this, using a top .sdoc with grammar parent ref optional. Including a specific requirement titled “derived” on which all other .sdoc derived reqd will point as parent ref. But this might be improved.

Children:

- [SDOC-BACKLOG-9] *Derived requirements*

13. Technical Note: Zephyr requirements tool requirements

13.1 Multiple files / include mechanism

UID:	ZEP-1
STATUS:	Active

Requirements or groups of requirements shall be distributable over several files and still form a full specification (document) via some kind of include mechanism.

RATIONALE:

In a future constellation the requirements shall be written resp. update with the code in the same PR. Smallish requirements files per topic / component next to the code in the same repo allow a better workflow than one huge requirements file somewhere.

Children:

- `[SDOC-SSS-52]` *Assembling documents from fragments*

13.2 Clear separation of requirements (machine-readable)

UID:	ZEP-2
STATUS:	Active

Requirements objects shall be clearly separated from each other, also when organized in the same file.

RATIONALE:

For exporting or machine processing, a clear separation of requirements objects is a prerequisite.

Children:

- `[SDOC-SSS-54]` *Machine-readable format*

13.3 Custom fields

UID:	ZEP-3
STATUS:	Active

Requiremements objects shall be configurable to create several types with a number of custom fields.

RATIONALE:

Requirements on software level may need to hold different information than on the architecture/interface and on the component level. By having typed requirements objects, linkages between requirements objects can be verified and filtered (start_object_type – link_role_type –> end_object_type)".

Children:

- [SDOC-SSS-62] *Custom fields*

13.4 Links

UID:	ZEP-4
STATUS:	Active

Linking shall in general be supported between any requirement object of any object type in a 1:n manner.

RATIONALE:

A SAIS requirement will link to a SRS requirement via «refines» link. A SITS test case will link to the same SAIS requirement.

Children:

- [SDOC-SSS-7] *Link requirements together*

13.5 Multiple link roles

UID:	ZEP-5
STATUS:	Active

Links shall be configurable to create multiple link roles.

RATIONALE:

Link roles and requirements object types allow to verify, that the traceability is consistent.

Children:

- [SDOC-SSS-8] *Multiple link roles*

13.6 ReqIF export

UID:	ZEP-6
STATUS:	Active

Requirements specification shall be exportable to ReqIF.

RATIONALE:

Will/may be used to as exchange format to generate a requirements and traceability documentation.

Children:

- [SDOC-SSS-58] *ReqIF export/import*

13.7 CSV

UID:	ZEP-7
STATUS:	Active

Requirements specification shall be exportable to CSV.

RATIONALE:

Will/may be used to as exchange format to generate a requirements and traceability documentation.

Children:

- `[SDOC-SSS-59]` *CSV export/import*

13.8 Unique ID management

UID:	ZEP-8
STATUS:	Active

Requirements objects shall allow unique ID management when adding new requirements on different branches.

Options could be:

- UUID: no checking required, but not handy
- Manually assigned: collision checking required
- Centralized: when not affected by branching".

RATIONALE:

Centralized object ID management might collide with a branching, PR, merging process approach commonly used in the rest of the project.

Children:

- `[SDOC-SSS-6]` *Auto-provision of Requirement UIDs*

13.9 Text formatting capabilities

UID:	ZEP-9
STATUS:	Active

The description field shall allow for formatting such as:

- lists
- tables
- headings
- UML diagrams
- etc.

RATIONALE:

In some cases a plain text requirement is not sufficiently clear and requires formatting or even UML diagrams.

Children:

- [SDOC-SSS-63] *Text formatting capabilities*

13.10 Minimal requirement field set

UID:	ZEP-10
STATUS:	Active

A requirements object shall at least comprise the following fields (or similar):

- title
- ID
- Description
- Status
- Outbound links
- Inbound links (optional?)

RATIONALE:

TBD

Children:

- [SDOC-SSS-61] *Minimal requirement field set*

13.11 Requirements to source code traceability

UID:	ZEP-11
STATUS:	Active

Linking from requirements objects to code or from code to requirements objects via ID shall be supported.

RATIONALE:

For safety development and certification linking to code is required.

Children:

- [SDOC-SSS-72] *Traceability between requirements and source code*

13.12 Non-intrusive links in source code

UID:	ZEP-12
STATUS:	Active

Linking from code to requirements objects via ID shall be least code intrusive.

RATIONALE:

Code with lots of meta information in it via comment tags, makes the code less readable. Links should best be hidden in existing comment structures e.g. function headers and not be extra tags.

Children:

- [SDOC-SSS-72] *Traceability between requirements and source code*

13.13 Structuring requirements in documents

UID:	ZEP-13
STATUS:	Active

Requirements objects shall be structurable in a document like manner (with requirements ordering, and organized in chapters).

RATIONALE:

A collection of unorganized requirements as a specifications are hard to read and understand. They should be organizable in topic chapters or similar.

Children:

- [SDOC-SSS-64] *Structuring requirements in documents*

13.14 Status field

UID:	ZEP-14
STATUS:	Active

Each requirements object type shall have a configurable status workflow.

RATIONALE:

Requirements may be in different statuses such as Draft, InReview, Approved. Dependent on the used process is rather reflected in the development work (branch=draft, PR under Review=InReview, PR merged to main=Approved.

Children:

- [SDOC-SSS-61] *Minimal requirement field set*

13.15 Tool Qualifiability

UID:	ZEP-15
STATUS:	Active

The Requirement Tool shall be qualifiable for use in safety-related and/or security-related development. At the very least, the Requirement Tool shall come with its own set of requirements, which shall be amenable to validation in compliance with the relevant standards.

RATIONALE:

Certification of Zephyr-based products.

Children:

- `[SDOC-SSS-50]` *Self-hosted requirements*