Documentation Guide

ezRead Documentation System

Version 2 Release 66

June 22, 2016

Comments & questions
Helmut Scherzer

Giesecke & Devrient
Prinzregentenstr. 159
D-81677 Munich

Fon.: +49 89 4119 2084
Mob: +49 174 313 9891

helmut.scherzer@gi-de.com
www.gi-de.com

Content

1 About this book 7
1.1 Quick start 7
1.1.1 "I want to install the ND.API" 7
1.1.2 I want to use instant links 7
1.1.3 I want to set the project environment 7
1.1.4 I want to generate named destinations 7
1.1.5 I want to work with MS-WORD 7
1.1.6 I want to understand the ND.API 7
1.1.7 I want to send linked documents to another person 7
1.1.8 I want to create the evironment for instant links 8
1.2 What this book wants 8
1.3 Acknowledgements 9

2 Documentation Architecture 11
2.1 About documentation architecture 11
2.2 Roles and stakeholders 11
2.2.1 Customer 12
2.2.2 Product Manager 12
2.2.3 G&D Project Management 12
2.2.4 System- / Software Architects 12
2.2.5 Development Team 13
2.2.6 Certification Group 13
2.3 Documents overview 15
2.3.1 Customer Requirements Specification (CRS) 16
2.3.2 Customer initiated Change Requests (cCRQ) 16
2.3.3 Stakeholder Requirement Specification (STARS) 17
2.3.4 System Requirement Specification (SysRS) 17
2.3.5 Architecture documentation 18
2.3.6 FSC - Component Specifications 18
2.3.7 STD - Standardization documents 19
2.3.8 Legacy documents 19
2.4 Processing the documents 19
2.4.1 Requirements management 19
2.4.2 DOORS 20
2.5 Product documentation deliverables 20

3 About ezRead 21
3.1 Requirements for documentation excellence 21
3.1.1 Req01: Intuitive reading 21
3.1.2 Req02: Easy to use 22
3.1.3 Req03: Consolidated information 23
3.1.4 Req04: Accurate and Up-To-Date 23
3.1.5 Req 05: Maintainability 23
3.1.6 Req06: Consistent document design 24
3.1.7 Req07: Media correctness and easiness of export 24
3.1.8 Req08: Modularity 24
3.2 How to verify that requirements are met 25
3.2.1 Formal criteria 25
3.2.2 You Only Know it's "Good Enough" when your readers tell you so! 25
3.3 ezRead Documentation Strategy 26
3.3.1 Using books 26
3.3.2 Cross-Referencing aspects 26
3.3.3 Editorial Team 27
3.3.4 Documentation rules 28
3.3.5 Meta Tags Strategy 28
3.3.6 PDF export format 29

3.4 Questions and answers . . . 30
3.4.1 Why not a WiKi? . . . 30
3.4.2 Why using a selected editor's team? . . . 30
3.4.3 Why asking for high quality documentation at all? . . . 31
3.4.4 Why don't we have such documentation today? . . . 31

4 Reading ezRead Documents . . . 35
4.1 Book organization . . . 35
4.2 Architecture documents . . . 36
4.2.1 Variants . . . 36
4.2.2 REQ - The book of requirements . . . 36
4.2.3 ASY - System architecture . . . 38
4.2.4 ASW - Software architecture . . . 39
4.2.5 AHW - Hardware architecture . . . 40
4.2.6 IMP - Implementation documentation . . . 40
4.2.7 TST - Test architecture . . . 43
4.2.8 CRT - Certification documents . . . 43
4.3 Accompanying documentation . . . 44
4.3.1 Storyboard . . . 44
4.3.2 Documentation Guide . . . 44
4.3.3 Coding conventions . . . 44
4.4 Reference documentation . . . 45
4.5 Project documentation . . . 45
4.6 Tips and hints for reading ezRead . . . 47
4.6.1 Notations in ezRead documentation . . . 47
4.6.2 Working with Adobe Reader . . . 47
4.6.3 How to navigate with Adobe Reader . . . 47

5 Writing Architecture documents . . . 51
5.1 Main Concepts . . . 51
5.1.1 Use a structure template for documentation . . . 52
5.1.2 Architectural views to separate concerns . . . 52
5.1.3 Top-Down approach to manage complexity . . . 52
5.1.4 Blackboxes to describe responsibility and interfaces . . . 53
5.2 Architecture documentation structure overview . . . 54

6 Writing ezRead Documents . . . 63
6.1 Style recommendations . . . 64
6.1.1 Using standardized paragraph styles . . . 64
6.1.2 Using highlighting techniques . . . 64
6.1.3 Using hyperlinks (Cross-references) . . . 64
6.2 Using Graphics . . . 66
6.2.1 Include time stamps to architectural images . . . 66
6.2.2 Using VISIO graphics . . . 66
6.2.3 Using Image import . . . 66
6.3 Using formatting templates . . . 67
6.3.1 The FormatRef.FM file . . . 67
6.3.2 The SysVars.FM file . . . 69
6.3.3 Don't care for document layouts . . . 69
6.4 Where to put documents . . . 70
6.5 Using meta-tags . . . 72
6.5.1 <xref . . . 72
6.5.2 <v . . . 74
6.5.3 <url . . . 74
6.5.4 <fig . . . 75
6.5.5 <tab . . . 76
6.5.6 <eq . . . 77
6.5.7 <ix . . . 77
6.5.8 <ND . . . 78
6.5.9 <xmp>, <exmp> . . . 78

6.6 Preparing books for the final release 80
6.7 Version management 81
6.7.1 Static targets 81
6.7.2 Dynamic targets 81
6.7.3 Target tagging techniques 81
6.7.4 Chapter number oriented referencing 82
6.7.5 Named Tag oriented referencing 82
6.7.6 FrameMaker referencing 82
6.7.7 Best practices to overcome the ‘moving target’-Problem 82
6.7.8 Content oriented version management 83

7 Cross Reference Techniques 85
7.1 Using Cross References and hyperlinks 85
7.1.1 Cross References 85
7.1.2 Hyperlinks to named destinations 85
7.1.3 How to view named destinations in a PDF file 86
7.1.4 How to create a named destination manually 88
7.1.5 How to create named destinations automatically 92
7.1.6 Stub technique 101
7.1.7 Content based destination generation 102
7.1.8 How to enumerate bookmarks 105
7.1.9 Enumerate – Merge function 107
7.1.10 How to jump to a named destination 107
7.1.11 Linking FrameMaker to SECURED documents 108
7.1.12 Instant Jump to Stub 109
7.1.13 Instant Jump with dialog box 109
7.2 Exporting books and references 110
7.2.1 Using the ND.API Export function 110
7.2.2 Target directories 112
7.2.3 References from Books to Books 114
7.2.4 Missing references 115
7.2.5 Clearing the export directory 116
7.3 Regular expressions 116

8 Working with MicroSoft Office (MS-WORD) 119
8.1 Linking files with MS-WORD 120
8.1.1 Linking any external document 120
8.1.2 Link to external MS-WORD document 120
8.1.3 Link to external PDF with named destination 121
8.1.4 Link to external MS-WORD document with DOC bookmark 122
8.1.5 Using MS-WORD bookmarks 123
8.2 Printing MS-WORD document as PDF 124
8.2.1 Using Bookmarks from MS-WORD 126
8.3 Unwanted artifacts in MS-WORD PDF conversion 128
8.3.1 *DOC* file links are not converted to *PDF* targets 128
8.3.2 MS-WORD creates URLs as links 128
8.3.3 MS-Word produces unstable links 130
8.3.4 Office 2003 does not take in bookmarks straight forward 130
8.3.5 Office 2003 does not allow straight ND 131
8.4 Converting MS-WORD links 132
8.5 Use case - from DOC2DOC to PDF2PDF 133
8.5.1 1. Write target with bookmarks 134
8.5.2 2. Converting to PDF with WORD bookmarks 134
8.5.3 3. Generate Named Destinations from Bookmarks 137
8.5.4 4. Generate Stubs 137
8.5.5 5. Link Stubs 138
8.5.6 6. Converting source file to PDF 138
8.5.7 7. Convert MS-links 138
8.6 Creating bookmarks from comments 138

9 Storing Reference Documents . . . 139
9.1 Where to store reference documents . . . 139
9.2 Document naming conventions . . . 139
9.2.1 Using separate convention profiles . . . 140
9.2.2 Reference Document File Name Format . . . 140
9.2.3 Examples for document naming convention . . . 142
9.2.4 Document Prefix for typical product documents . . . 142
9.2.5 Typical workflow order . . . 144
9.3 ezRead bibliography naming conventions . . . 144
9.3.1 Scientific [n] notation . . . 144
9.3.2 [Abbrev] notation . . . 144
9.3.3 [ezRead#9.3.3] notation and syntax . . . 145

10 Verifying documentation consistency . . . 147
10.1 Contents and Scope of this Document . . . 147
10.2 Priorities . . . 147
10.3 Acceptance Criteria for Architecture Documentation . . . 149
10.3.1 Formal Criteria . . . 149
10.3.2 Criteria for understandability of documentation . . . 150
10.3.3 Criteria for architecture goals and constraints . . . 151
10.3.4 Criteria for Context View . . . 152
10.3.5 Criteria for building blocks view . . . 152
10.3.6 Criteria for Runtime View . . . 154
10.3.7 Criteria for deployment view . . . 155
10.3.8 Criteria for Design Decisions . . . 156
10.4 Acceptance criteria for architecture . . . 157
10.4.1 Related to product quality metrics . . . 157
10.5 Acceptance criteria for development . . . 158
10.5.1 Source code artifacts . . . 158
10.5.2 General coding conventions . . . 158
10.5.3 Related to JAVA coding conventions and styleguides . . . 159
10.5.4 Related to Cocoon & XSLT Coding conventions and styleguides . . . 161
10.5.5 Criteria related to Version Control . . . 161

11 Installing ezRead accompanying Tools . . . 163
11.1 Who should install? . . . 163
11.2 How to setup the documentation system . . . 163
11.2.1 Installation . . . 164
11.2.2 Removal . . . 164
11.3 Manual Installation . . . 164
11.3.1 Associating stub's .STB extension manually . . . 164
11.3.2 Associating to AcroDDE.EXE . . . 171
11.3.3 Changing Adobe Reader Security Settings . . . 174
11.4 Installing with 01_InstNd.EXE . . . 175
11.5 Removing the installation . . . 175
11.6 Installing Fonts . . . 176
11.6.1 Using standard font installation . . . 176
11.7 Set/Clear documentation environment . . . 178
11.7.1 Setting the documentation environment . . . 178
11.7.2 Clearing the documentation environment . . . 180
11.7.3 Setting the documentation environment from command line . . . 180
11.7.4 Reading the present documentation environment . . . 181

12 Technical Guide . . . 183
12.1 The ND.API plug-in . . . 183
12.1.1 Navigate Documents . . . 184
12.1.2 Set Named Destination . . . 185
12.1.3 Export/Import named destinations . . . 185
12.1.4 NDX:Named destinations - CSV File structure . . . 185
12.1.5 Clear named destinations . . . 187

12.1.6 Generate named destinations from bookmarks 188
12.1.7 Generate bookmarks from named destinations 188
12.1.8 Generate named destinations/bookmarks from comments 191
12.1.9 Get named Destination from selected Text range 194
12.1.10 Export/Import Bookmarks 194
12.1.11 Enumerate bookmarks 196
12.1.12 Focus selected Bookmarks 196
12.1.13 Generating stubs from Bookmarks 197
12.1.14 Generating Stubs from named destinations 198
12.1.15 Generate Stubs from Dest with coordinates 198
12.1.16 Setting Bookmark Stubs manually 199
12.1.17 Bulk generation of stubs from document title 199
12.1.18 Stub management 200
12.1.19 Create Hyperlink from selected text 200
12.1.20 Convert links 204
12.1.21 Set/Clear Documentation Evironment 206
12.1.22 File/Path Function 206
12.1.23 Set ND view reference 207
12.1.24 Fix current Zoom 208
12.1.25 Generate Slides 208
12.1.26 Exporting linked files 209
12.1.27 Clearing export directory 209
12.1.28 Remove Personalization 209
12.2 Accompanying tools 214
12.2.1 Toc2Bkx.EXE 214
12.2.2 Bkx2Stb.EXE 214
12.2.3 SetEnvReg.EXE 214
12.2.4 AcroDDE.EXE 214
12.2.5 01_InstND.EXE 216
12.2.6 02_AutoHotkey_L_Install.exe 216
12.2.7 ClipText.EXE 218
12.2.8 ClipJoin.EXE 218

13 Glossary and Abbreviations 219
13.1 Abbreviations 219
13.2 Glossary 220

14 INDEX 225

1 About this book

1.1 Quick start 7
1.2 What this book wants 8
1.3 Acknowledgements 9

1.1 Quick start

The following is a quick start instruction for users who plan a particular task to do. This entire book will describe many aspects of the ezRead system and the ND.API hence it is good styls to allow the reader find the most important places right from here.

The steps are shown in the right order of processing if you want to install the ezRead system and do the first steps.

1.1.1 "I want to install the ND.API"

→ 11.2 “How to setup the documentation system”

1.1.2 I want to use instant links

Precondition Someone created the stubs in **X:** → 12.1.14 “Generating Stubs from named destinations”

Installation

1. Install AcroDDE.EXE → 11.4 “Installing with 01_InstNd.EXE”
2. Install AutoHotkey → 12.2.6 “02_AutoHotkey_L_Install.exe”
3. Apply the Basic script in AutoHotkey → 12.2.6.1 “Basic HotKey script – 03_Basic.ahk”
4. Set Project Environment to "X:\Documents → 11.7.3 “Setting the documentation environment from command line”
5. Test the system → 7.1.12 “Instant Jump to Stub”

1.1.3 I want to set the project environment

→ 11.7 “Set/Clear documentation environment”

1.1.4 I want to generate named destinations

→ 7.1.5 “How to create named destinations automatically”

1.1.5 I want to work with MS-WORD

→ 8 “Working with MicroSoft Office (MS-WORD)”

1.1.6 I want to understand the ND.API

→ 12.1 “The ND.API plug-in”

1.1.7 I want to send linked documents to another person

→ 7.2.1 “Using the ND.API Export function”

1.1.8 I want to create the evironment for instant links

→ 12.1.14 “Generating Stubs from named destinations”
→ 7.1.12 “Instant Jump to Stub”

1.2 What this book wants

This book is describes the ezRead documentation architecture used for the technical documentation in selected G&D projects and in partner relationships.

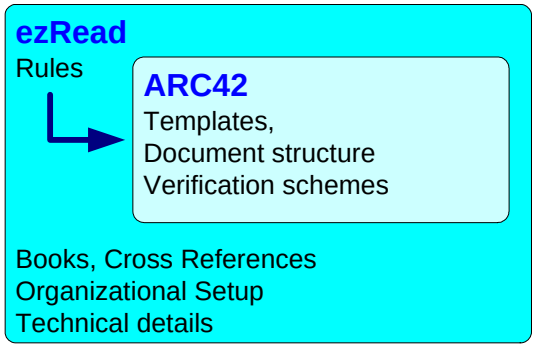


Figure 1-1. ezRead system integrates ARC42

ezRead is a name created by G&D. It is built from the idea “Easy Read” using popular American shortage (e-z-Read).

This book provides an introduction to the software and system architecture documentation standard applied for (a revised and customized version of the ARC42 architecture documentation template).

Acceptance criteria for architecture and architecture documentation, which shall be met by future releases of the architecture and the respective architecture and design documents, are described in a separate chapter 10.

Chapter 2 “Documentation Architecture” on page 2-11 provides general information on the typical roles and deliverables associated to a product development. It also provides important definitions of document types used throughout the rest of the book.

Chapter 3 “About ezRead” on page 3-21 describes the rationale for the *7-book documentation strategy* and discusses several questions regarding the pro- and cons of comparable approaches.

Chapter 4 “Reading ezRead Documents” on page 4-35 explains how the ezRead technique is used from the reader’s point of view

Chapter 5 “Writing Architecture documents” on page 5-51 explains how the ezRead technique is used

Chapter 6 “Writing ezRead Documents” on page 6-63 instructs writers of technical documentation on document structure and recommended rules for product documentation.

Chapter 7 “Cross Reference Techniques” on page 7-85 describes the methods and variants of cross referencing, as much of the elegance of ezRead comes from cross-referencing, this separate chapter focuses directly to this aspect.

Chapter 8 “Working with MicroSoft Office (MS-WORD)” makes an important discussion about the work with Microsoft Office versions, reflecting the latest release Office 2010 and also artifacts with prior releases until Office 2003. Important for editors working with Office products.

Chapter 9 “Storing Reference Documents” on page 9-139 discusses where to store reference documents and the related naming convention.

Chapter 10 “Verifying documentation consistency” on page 10-147 describes criteria to verify the conformance of architecture documents with the rules of 6 “Writing ezRead Documents”.

Chapter 11 “Installing ezRead accompanying Tools” on page 11-163 describes the installation processes and ezRead configuration settings

Chapter 12 “Technical Guide” on page 12-183 describes the technical aspects of ezRead processes and tools.

1.3 Acknowledgements

My special thanks go to David C Toll, a former collegue and present friend in the IBM Research Division of the T.J. Watson Research Center in New York. Over many years of contigous effort David has created and maintained “on the side” an excellent user guide to describe techniques and features of the Adobe FrameMaker product usage and many tips and hits around professional documentation.

David has granted G&D - in the name of IBM - free usage of his precious book and I did not hesitate to refer to its contents wherever it is helpful for this manual. For editorial teams using FrameMaker his document is a “must have”.

(→ Generating BookMaster Style Documents using Adobe® FrameMaker® 7.1)

Helmut Scherzer, Munich, April 20nd 2012

2 Documentation Architecture

2.1 About documentation architecture 11
2.2 Roles and stakeholders 11
2.2.1 Customer 12
2.2.2 Product Manager 12
2.2.3 G&D Project Management 12
2.2.4 System- / Software Architects. 12
2.2.5 Development Team. 13
2.2.6 Certification Group 13
2.3 Documents overview 15
2.3.1 Customer Requirements Specification (CRS). 16
2.3.2 Customer initiated Change Requests (cCRQ) 16
2.3.3 Stakeholder Requirement Specification (STARS). 17
2.3.4 System Requirement Specification (SysRS). 17
2.3.5 Architecture documentation 18
2.3.6 FSC - Component Specifications 18
2.4 Processing the documents 19
2.4.1 Requirements management 19
2.4.2 DOORS. 20
2.5 Product documentation deliverables 20

2.1 About documentation architecture

This and the following chapters describe a general documentation strategy "ezRead" for complex products. ezRead is a name creation expressing the emphasis on the reader's view on good documentation. It is made from popular American style associating "easy reading" (= e - z - Read). The name is not a standardized term. ezRead however, does relate to public proposals like ARC42 and enhances them by several ideas.

ezRead focusses on *internal product related documentation*. The main motivation of the documentation strategy is, to define a set of *readable* (no kidding!) documents with high complex technical content. *Readability* allows an *intuitive* and *easy-to-follow* approach. Due to reasons given below, today, most internal product description little does maintain those aspects

ezRead is nothing radically new, nor does it change things upside down. It is a lean and efficient write-up of ideas that have been practised by good technical editors for times. Maybe the biggest quality of ezRead is, that it is all written up.

Documentation is found in many places in a project. In general we distinguish between Project documentation and Architecture documents. Today requirements management systems (e.g. DOORS) are used to cover the demands of complex products. The next three chapters describe the typical development process from Customer Requirements Specification (CRS) to the final certified product.

2.2 Roles and stakeholders

Roles

The typical roles associated to document management are

- Customer (Product-/ Project Management)
- G&D Product Manager
- G&D Project Management (Development)
- System- / Software Architects (Requirements Engineers Architects)
- Development Team (Coding / Test / Certification)

Every role contributes to the final documentation batch in a particular way. In the following a rough understanding is provided for the most prominent duties of the

2.2.1 Customer

With respect to the requirements management the customer will involve technical experts to

- provide the list of customer requirements → CRS
- workout STARS together with G&D
- approve STARS to consolidate the covered requirements
- manage Late-In-Time change requests
- verify intermediate results during project development.

2.2.2 Product Manager

The Product Manager is responsible to maintain active communication to the customer. With respect to the requirements management the product manageer shall

- understand the main requirements to monitor their fulfilment throughout the project
- connect customer's experts with G&D's experts
- coordinate documentation flow both ways and negotiate the access credentials to documents (both ways)
- organize early feedback from customer in order to avoid misalignment in the understanding of requirements
- manage customer change requests (cCRQs) or late-in-time documents.
- prepare the approval of the STARS specifications

2.2.3 G&D Project Management

The complexity described in Figure 2-1 is an indicator how much project management needs to be involved to organize the right pace and activities to progress with multiple threats.

Also project management is not directly involved into the creation and workflow related to requirements, architecture and software development, knowledge about the documentation tree and project control is essential to keep risk and cost low and efficiency high. In particular the provision of resources, organization of agile teams (optional) and the milestone management is important to avoid bottlenecks in the process described in 2.4 "Processing the documents".

2.2.4 System- / Software Architects

The architecture team very often copes with two missions, once the system requirements engineering and on the other side the system architecture. Both of them are well associated and the close relationship between these disciplines is an efficient way to achieve high product quality.

The main tasks of requirements engineering are:

- consulting the customer and joint effort to achieve a comprehensive STARS which can be approved on time
- *correct interpretation and classification of the customer's requirements*, derived from the Customer Requirements Specification (CRS). Customer reqirements may be received in perfect organized form or as a collection of (sometimes contradicting) claims in a bunch of documents, mixed up between legal, organizational and technical

statements. The correct interpretation of the requirements and dispatching of the different levels is essential for the correct fulfillment in the final product. Contraditions in the CRS need to be sorted out prior to STARS approval.

- elaborate comprehensive set of system requirements after the STARS is approved.
- maintain a set of system requirements corresponding to the architecture development

The main tasks of system architects are:

- create efficient building blocks from the system requirements
- create efficient interfaces between the building blocks and to external components
- provide stacked models (typ. UML) to refine the views on several functional blocks
- stimulate the architectural concept with associated scenarios in order to validate the intended functionality.
- define architectural constraints and conditions
- respect platform and standard references
- define system requirements that result from architectural design

2.2.5 Development Team

The development is typically represented by two main groups, coding engineers and test engineers. In a typical project the share is around 40 (test):60(coding).

Both, coding and testing share the following tastks

- take up architecture and system requirements to generated code/tests.
- create FSC - Component Specifications for code if need shall arise
- document coding/testing often realized by auto-documentation tools that convert comments into accompanying docu (doxygen, Javadoc, Autodoc and other tools → 4.2.6.1 “Using auto-generated documents” on page 4-41
- provide references to architecture and system requirements where appropriate
- make reuse of existing modules to improve efficieny and stability of generate code/test
- provide proper modularity in coding/testing to allow reuse of modules in later projects.
- respect particular security techniques that may not be explicitly exposed in architecture, but known as best coding/testing practise.
- create baselines (releases) of intermediate code/test in order to allow iterative progress.

Other legal obligations like site restrictions and data retension are well known, but secondary to this document’s focus on requirements and architecture documentation.

2.2.6 Certification Group

The certification group provides evaluation documents according to the designated security target. Depending on the planned evaluation level and type (CC vs. type approval etc.) the group can be made of few or only one person only. That person, however, needs to be expert in evaluation and typically has an appropriate qualification.

Certifification starts late, typically if the product is close to be finalized. The actual evaluation can only be done on the final code, since it is only the one code to be certified after all. The main task of the certification group are

- Prepare the product (code/test) such that completeness can be verified
- Determine the set of documents required for the associated security target
- Create evaluation documents, partly assisted by development
- Scrutinize the code for all instances that satisfy the claims of the security target
- Build links from the evaluation claims to the documents that describe the solution, those documents may be Architecture documentation and FSC - Component Specifications and of course the code itself. Requirements may be a good guide to the implementation, but certification investigates the actual response (=implementation) to those requirements.
- Cooperate with external evaluator to provide the necessary information and correct layout of documents.

2.3 Documents overview

Figure 2-1 shows a representative example for a project's documentation landscape. The figure is to be read counterclockwise starting with the top right corner where it all begins with the customer's specifications (CRS).

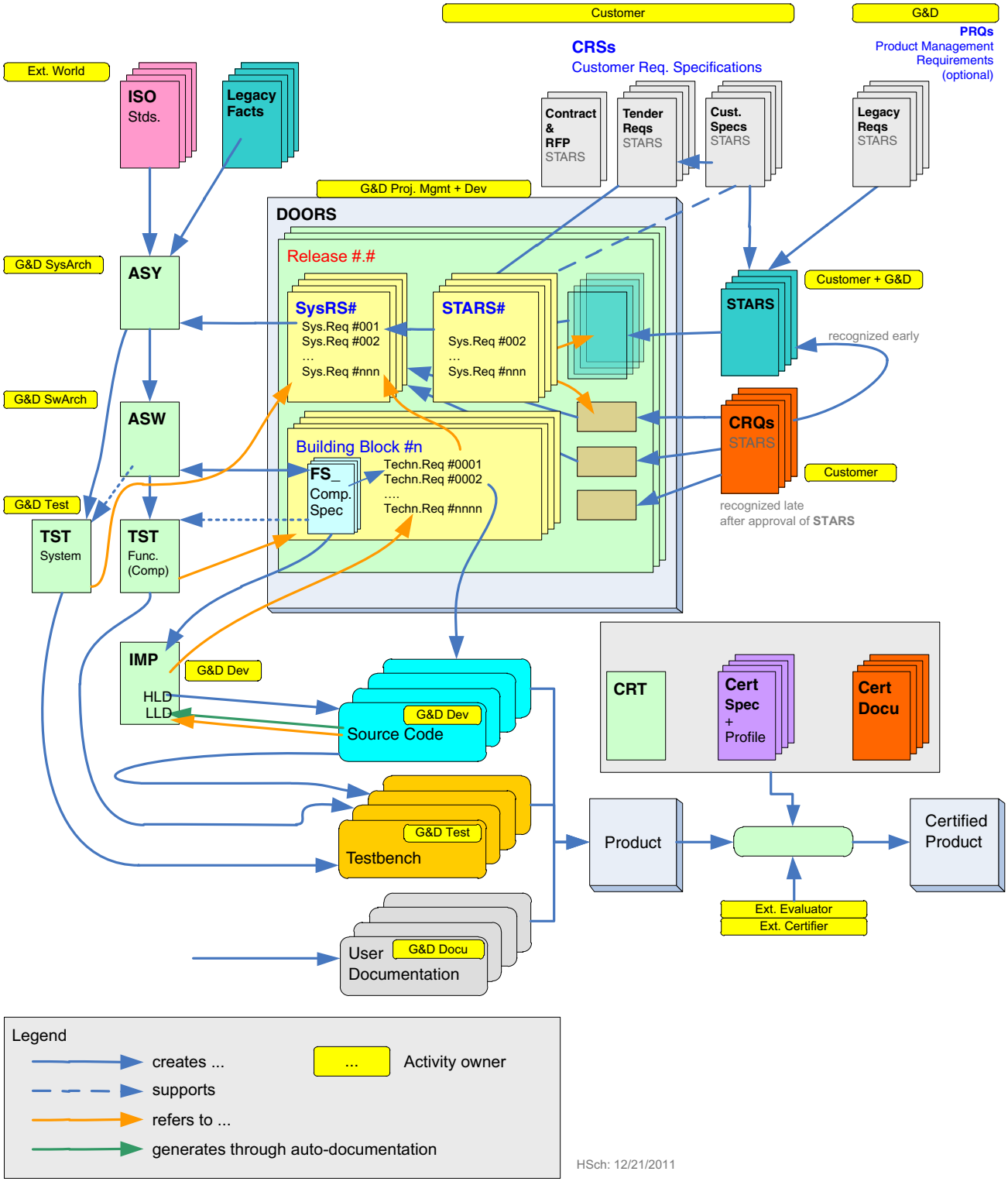


Figure 2-1. Example of detailed view on a project requirements

2.3.1 Customer Requirements Specification (CRS)

Project flow

Reading from the top right position, the customer creates the Customer Requirements Specifications (CRS). Depending on the product, these specifications are a collection of existing material (if a previous product did exist) and a set of new requirements explicitly written by the customer for the specific project.

Customer documents

Customer documents that can carry CRS requirements implicitly are

- *Contract documents* that refer to a set of specifications
- *Tender documents* that reflect the scope of the negotiated work and product, often by numbered requirements.
- Explicit customer specifications that define more details of the final product.

CRS content

In general it is not known at which level the CRS are delivered. Possible levels are

- *Brief collection* of requirements, little specific claims
- *STARS* level, specific requirements to product behaviour
- *SysRS* level, specific requirements to product function and interfaces
- *Architectural documents*, if (existing) architecture shall be suggested by the customer
- *Component Specification*, detailed functional specification according to which the product can be built and tested.
- In large government or payment projects it is possible that the customer provides a *full set of approved component specifications* which were created by an external third party. This is common practise for national rollouts and large systems in which several vendors are ordered concurrently.

Back referencing requirements

The final goal to be achieved is the correct extraction and sorting of all requirements that the customer has adressed. Very often it is reasonable to allow back-referencing of requirements to the places where the customer had made the claims.

Static system requirements

In particular, if the customer provides specifications at the level of FSC - Component Specifications, the claims made in this specifications are relevant for all levels from *STARS, SysRS, Architecture documentation* until the implementation. For each of these levels the claims of such a document are integrated as "static requirement" . This method allows the customer to introduce claims at a very detailed level.

2.3.2 Customer initiated Change Requests (cCRQ)

A customer change request is any customer initiated demand for change in the requirements after the date of the approved STARS. Two flavours of CRQs are relevant

Fixing design

Fix CRQs: These are change requests that fix a problem in the previous design which was not recognized before. The acceptance of such change is mandatory to provide proper function of the product. There shall not be any additional charge to the customer if he orders such change request.

Adding functions

Feature CRQs: A customer initiated feature CRQ enhances or improves the function of the final product. Depending on its impact, such CRQ can be migrated easily or at high expense. If for instance a customer wants to enhance the functionality of a DVD-Drive to BluRay, then the entire architecture may be put in question.

Pay CRQs ?

Typically the customer is charged for Feature CRQs, the actual business aspects, however, are subject of the associated business contract.

2.3.3 Stakeholder Requirement Specification (STARS)

This specification collects the minimum number of statements required to measure whether the product was built according to the customer's expectation or not.

Neither the quality nor the content distribution of any customer documents (where are the requirements ?) are predictable. Therefore the initial work to be done is the collection of requirements into a STARS. This will often be done in cooperation with the customer.

The Stakeholder Requirement Specification (STARS) covers all requirements to describe the general behaviour of the final product from the customer point of view. In contrast to a System Requirement Specification (SysRS) the STARS may be more relaxed, but it shall cover all necessary requirements to allow the customer an approval of the requirements.

Example: A STARS specification may claim resistance to EAL.VAN4. An associated SysRS needs to specify which attacks are to be covered explicitly. An architecture willl suggest the appropriate countermeasures whereas the (optional) component specification would suggest specific implementation details for the countermeasure.

Finally the STARS is subject to approval between customer and provider. The scope of the STARS therefore needs to be comprehensive and it should cover all claims that were negotiated on a business level agreement (e.g. contract). Very often - signing the STARS is associated to a project milestone.

Tools

Typically the STARS will be realized with the DOORS tool.

2.3.4 System Requirement Specification (SysRS)

This specification collects the comprehensive number of statements relevant to solve the claims of the Stakeholder Requirement Specification (STARS) from the view of the architecture.

A requirement in the STARS may well put out further requirements in the SysRS to describe the problem to be solved in an understandable way.

Example: A STARS requirement may claim: "A command length of 2 KB shall be supported". The associated sub-claims of a SysRS would specify:

1. "The extended APDUs as specified in ISO 7816-4 shall be supported."
2. "For T=0, the ENVELOPE commands need to be implemented
3. "For T=1, data blocks larger than 256 bytes will be transported through transmission-chaining (in contrast to command chaining on application level).

It can be seen, that the three sub-claims are a direct consequence of the STARS requirement. Therefore it is common practise to specify system requirements as subset of stakeholder requirements.

System requirements do not only enhance existing stakeholder requirements but they may add up new functionality in order to achieve the associated effect.

> **Example**: A server system may claim 24/7@1000/s availability on STARS level. Without anticipating architectural design, the related SysRS statements would precise the claim by "Decentralized processing required".as this is a known consequence from the 24/7 claim. To process 1000 requests per second, an additional system requirement may request a monitoring system to measure the number of incoming calls and a dispatcher unit to route the calls to available servers. That last claim, however, is not a direct demand from the STARS requirement but is additional in the SysRS to work out the solution of the 24/7@1000.

Since System requirements are closely related to architecture, they often work hand-in-hand with architectural design. Iterative development models (Agile/Scrum) may be applied to accelerate the elimination of inefficient design.

Tools

Typically the SysRS will be realized with the DOORS tool.

2.3.5 Architecture documentation

Architecture describes and prescribes the “*solution*” to the *requirements* and therefore it describes and guides its implementation. Architecture is an abstraction of sourcecode artifacts and does therefore not describe or prescribe implementation detail (but of the architecturaly relevant ones!).

The term *software design* can be used to describe implementation details: design = architecture-in-the small. In our terminology, design is concerned with the internal structure of the lowest-level building blocks (→ 5.1.4 “Blackboxes to describe responsibility and interfaces”, especially the elaboration of building blocks as shown in Figure 5-3). The design of those lower-level building blocks is usually delegated to single developers, they design and develop the code-structure for these building blocks.

In context the term „architecture“ comprises *software* and *system* architecture. It consists of all structures, components, their interfaces and relationships, which together solve the requirements. A particular separation is done in 5.2 “Architecture documentation structure overview”.

The focus of „architecture“ is mainly on achieving *non-functional requirements*, like maintainability, flexibility and understandability of the whole system, plus support for all functional requirements as well.

Find more details about architecture documentation in 4.2 “Architecture documents” on page 4-36.

2.3.6 FSC - Component Specifications

Architecture is intentionally not thought to be an exhaustive set of specifications (→ 2.3.5 “Architecture documentation” on page 2-18). It may be possible to start programming from the base of system requirements and architecture documentation.

In other situations, the architectural description (→ 5 “Writing Architecture documents” on page 5-51) delivers components and interfaces, however, details of these components/interfaces need to be specified in further detail.

This can be done in *Component Specifications*, also known as "*Functional Specifications*".

A component specification is a detailed description of function and interfaces of a particular component of the system. It may include specific remarks, definitions and as well further architectural elements (e.g. Sequence diagram) in order to specify a functionality as precise that a software programmer cannot fail the final functionality through open questions.

2.3.7 STD - Standardization documents

Very often, external standards need to be respected for product design. Typically these standards or features described in those standards are claimed in the STARS but latest in the SysRS. An example is the 7816-X series for SmartCards.

There are also industrial standards which may be respected. For JAVA based products the Global Platform Standards are very popular. For cryptographic operations the algorithms approved in national bodies algorithms catalogues are highly relevant.

Hence architecture needs to consider those standards into the design in order to provide compatibility in function and interfaces.

Standard's claims may be overruled by the customer's requirements formulized in the Customer Requirements Specification (CRS), however those exceptions shall be explicitly described in the STARS.

2.3.8 Legacy documents

Architecture shall also consider existing material. Therefore a set of legacy documents needs to be referred to cover existing architectural blocks. Reuse has the advantage of faster time to market and reliable and proven concept. It is sometimes overlooked that such legacy documentation needs to be integrated, possibly modified and maintained for every 'child'-project.

While reuse may be well made on SysRS level, it will not be explicitly visible at that point since a system requirement is independent from whether it ever existed before or not. If it is a requirement, it can be considered "timeless".

On architecture level, however, legacy/reuse methods are visible since the may provid a quick path to an efficient solution. As architecture is the answer to system requirements, the answer may be "reuse".

2.4 Processing the documents

2.4.1 Requirements management

The content of the STARS is cloned (converted) into the DOORS system and the *system requirements* are created. They are still different from the *technical requirements* because those cannot be determined until architecture has designed the *building blocks*.

As far as claims of *external specifications* can be recognized at the creation time of the STARS they will be taken right into the STARS. However, it is normal that a *detaillled view* on those specifications is not done earlier then during the *system/software-design*. Therefore there are system requirements on top of those noted in the STARS.

It is also the case, that the *customer has additional requests* called CRQ. Although the customer is typically paying for those additions, they will generate other *system requirements* along the STARS documentation.

After the system requirements are settled in DOORS, *system architecture* starts design of building blocks. More on this can be found in ASY - System architecture.

However, system architecture will not answer to every particular system requirement because it is not meant to do so. Instead the *result* of system architecture does indeed present a *design* that is an answer to the *entire set* of system requirements. As architecture should summarize and struture it shall not answer in a detailled manner. As such it is not resasonable to include system architecture into the DOORS system.

It is the role of ASW - Software architecture (building classes and internal interfaces) to give specific answers to the system requirements. Classes built in software architecture are not yet specified in detail (methods and properties) but in their functional *behaviour*.

It is up to the *functional specifications*, often written by developers, to finally answer the system requirements in compliance with the architecture. Therefore the functional specifications are held in DOORS again and will give a direct reference back to the *system requirements* in the form of *technical requirements* and *descriptions*.

2.4.2 DOORS

DOORS is a tool to organize requirements. Requirements are stored as database entries which buys all the rich features that databases allow. In particlar, requirements can be tagged and seperate views on requirements are possible. For instance, the customer requirements may be nested with system requirements such to allow better association.

Another way to associate STARS requirements to SysRS requirements is rich link capability that a database-system allows.

Finally DOORS allows an organized reuse of requirements so that the modularization of requirements may directly asssociate a reuse of architecture and code/test modules.

The known weakness of the DOORS system is the limited richness of publication quality. While the list of requirements can be compared to EXCEL entries, this is far away from the style a book or a documentation is written. For requirements, this effect is minor and the advantages of the database approach count more.

For the same reason, however, architecture documentation shall not be provided by the DOORS system. The arguments are listed in 4.2.2.1 “Working with requirements management systems (e.g. DOORS)” on page 4-37.

2.5 Product documentation deliverables

The actual *product delivery* typically consists of

- source code (optional, depends on customer contract)
- test scenarios (comprehensive test, depends on customer contract)
- user documentation
- Certification report (optional)

whereas the entire set of other documents is important to maintain, re-use and extend the actual product after its product release. Further maintenance is often done by a much smaller number of developers.

Project Transfer

It also happens that a product is transferred to a different department for maintenance and updates. All the three project parameters *time-quality-cost* will then directly depend on the documentation excellence met by that documentation which is *not* delivered to the customer.

3 About ezRead

3.1 Requirements for documentation excellence 21
3.1.1 Req01: Intuitive reading 21
3.1.2 Req02: Easy to use 22
3.1.3 Req03: Consolidated information 23
3.1.4 Req04: Accurate and Up-To-Date. 23
3.1.5 Req 05: Maintainability 23
3.1.6 Req06: Consistent document design 24
3.1.7 Req07: Media correctness and easiness of export. 24
3.1.8 Req08: Modularity. 24
3.2 How to verify that requirements are met 25
3.2.1 Formal criteria. 25
3.2.2 You Only Know it's "Good Enough" when your readers tell you so! 25
3.3 ezRead Documentation Strategy. 26
3.3.1 Using books 26
3.3.2 Cross-Referencing aspects 26
3.3.3 Editorial Team. 27
3.3.4 Documentation rules. 28
3.3.5 Meta Tags Strategy. 28
3.3.6 PDF export format 29
3.4 Questions and answers 30
3.4.1 Why not a WiKi? 30
3.4.2 Why using a selected editor's team? 30
3.4.3 Why asking for high quality documentation at all? 31
3.4.4 Why don't we have such documentation today? 31

3.1 Requirements for documentation excellence

This section outlines the requirements and goals that shall be met by *architecture documentation*.

These requirements are *quality* goals, that means achieving them means effecting compromises: Many quality goals negatively affect each other, so that a global optimization is rather difficult. For example, accuracy and up-to-dateness of any documentation has negative impact on maintenance cost (as it takes more effort to keep the document current). The most important quality goals for architecture documentation are:

- Maintainability
- Understandability
- Accuracy
- Up-To-Dateness
- Transparency
- Stakeholder-Orientation

The criteria for good documentation are measured by

3.1.1 Req01: Intuitive reading

A contracting entity needs to keep a thorough understanding of the architectural structure of, over regular releases This is surely the biggest challenge to authoring. *Understandability* of documentation is affected by (at least) the following factors.

Find the right entry point fast

Today, technical literature is mostly used as *reference* documentation. A reader should find an *entry point* for his question in a *short time*. Many projects make it difficult to find this point because valuable information is distributed over several places, systems, and authors.

Good reference documentation uses a *single point of entry* and fast access to the topic, realized by as well as by technical means (e.g. tools to scan the information) and content organization (structured layout).

“The book of the books” is one approach to use a master document which guides to the different topics of a multi-volume documentation. A documentation map in the beginning of each of those volumes could be another realization of thiw requirement.

Find more detail

A reader might need more detail to his question. Good documentation will cross-reference into details, helping a reader to reach the point where the answer of his question is described. The magic of good cross-referencing is to determine intuitive links guiding a reader to the expected information.

Get the big picture

The opposite of the desire for “more detail” happens, if a reader gets lost in detail and needs to understand the big picture of a topic. Cross referencing is again the solution to bring the reader from a particular point of detail to a higher level with less granularity, but better overview characteristics.

Ease of Navigation & Retrievability:

These are achieved by or relying on established, standardized and well-documented *structure*, so that both readers and writers know in advance where to find or file certain information.

Keeping an up-to-date index of keywords and topics

for quick retrieval of information (supported by a project or system glossary)

Clarity & Style

both of which are subjective matters.

All the above claims are the ‘magic’ of good documentation. ezRead suggests a scheme to make those aspects become feasible by an affordable effort.

Note: In a way, all the following requirements contribute to this first one. Therefore the name of the documentation strategy has been selected to reflect the first requirement of easy reading (= e - z - Read).

3.1.2 Req02: Easy to use

This requirement is close to “Req01: Intuitive reading” but focusses a most important aspect for acceptance. Cross references need to be done by *everyone* writing documentation. But if we would ask a developer to create a cross references like *“..\Specifications\Database\MySpec.PDF:page27:Chapter 3.8.9”* we would receive very few contributions, simply because a developer doesn’t have the time to resolve path and details. Good documentation shall allow quick and easy use of procedures to maintain acceptance with the documentation creators.

Note: ezRead solves the above problem using Meta Tags Strategy.

3.1.3 Req03: Consolidated information

While a product or a product feature undergoes *several changes*, in a good project, with every change, an appropriate description, requirement definition and change request etc. is created. The accumulation of change requests, however, does not contribute to the readability of a feature description. Good documentation consolidates the important facts from existing documents and maintains one view instead of a historical record of changes.

3.1.4 Req04: Accurate and Up-To-Date

During the development of a project it occurs, that ideas, decisions, and design is created in *different documents*. It often happens, that changes are actually processed through good tool chains, requirement systems and change requests etc. but still the actual content of a change doesn't find its way into product documentation. After some time, the product documentation lacks synchronicity with the actual product.

The problem here is, that those changes do well process along the *project* documentation and tools but miss to update the *product* documentation. (see "The mix-it-up problem").

Good documentation *synchronizes the product information* from project information hence keeping the product description up to date.

Information contained within the architecture documentation has to be accurate in the sense of "correct", so that stakeholders (developers, system- and software-architects, testers etc.) can rely on this information. *Accurateness* of documentation is affected by (at least) the following factors:

- Every fact given in the documentation is *currently correct*. All changes in the system shall be reflected in the documentation in a timely manner (when the change occurs). Every change has to be documented not later than release-time.
- All *important* facts are given in the documentation. For architecture that means „completeness" without „exhaustiveness". (→ 3.1.5 "Req 05: Maintainability").
- *Missing information* is marked as such. When crucial information is missing, both reasons and time of remedy shall be given.
- If similar or even the *same information is written in two or more places*, then cross references and index entries shall be provided to establish the relationship between those places. This will be very efficient to synchronize changes.
- *Changes in the system* (in its requirements, architecture and implementation) shall be reflected in the respective documentation. There shall be an *change history,* so that changes can be tracked with appropriate effort (this is a tradeoff between accuracy and maintainance-costs).

  Change history shall start with the *first internal release* (for review).

3.1.5 Req 05: Maintainability

Due to significant maintenance costs associated with large-scale documentation, architecture documentation aims at keeping maintenance effort in balance with understandability, accuracy and up-to-dateness. *Maintainability of documentation* is affected by (at least) the following factors:

- *Amount* of documentation (shorter docu is easier to maintain)
- *Redundancy* (repeating information sometimes helps readers to avoid jumping around in documents, but increases maintencance effort)

- Established standard structure: See 3.1.1 “Req01: Intuitive reading” on understandability.
- *Tools* used: Appropriate tool support can significantly reduce overhead (see 12 “Technical Guide” on page 12-183).

The following steps shall be taken to support maintainability:

- *Limit the amount of details* (e.g. do not duplicate information that can be easily taken from sourcecode). However, architecture documentation is no replacement for inline code documentation, but shall be supported by a best-practice approach in source-code documentation (→ 1 “Development Process” in volume 3 page 1-5).

  Only „architecturally relevant“ details shall be included.
- Stick to the *established structure*, so the „location“ of documentation changes are simple to determine.
- Consolidate information regularly (→ Req03: Consolidated information).

3.1.6 Req06: Consistent document design

While a large product has many author’s, the set of documents being produced will reflect the diversity of the author’s. This starts with writing style and ends with graphic formats (from PS to BMP) of embedded figures. Good documentation eliminates this diversity and presents information in a consistent design. The reader gets much better used “how to read” the documentation.

Consistent document design is achieved through a defined document structure as suggested in Chapter 5 “Writing Architecture documents” on page 5-51.

3.1.7 Req07: Media correctness and easiness of export

An important aspect of documentation comes from the aspect of document distribution. In particular when using cross references it is important to have a simple scheme which can be handled in many different environments. *Sophisticated directory structures* are a good thing during development, for document distribution, however, they are everything but desired.

Good documentation minimizes the complexity of file pathes and hence improves the handibility.

3.1.8 Req08: Modularity

Very often, product documentation can be re-used in other publications than product documentation only. An example could be the help file for a program and indeed one can often find even a 100% match between a user’s guide and a help system. (Examples: Adobe FrameMaker UserGuide and the offline-help system).

When it comes to use only *parts* of documentation, a good documentation philosophy should allow for *reuse* of components or at least for the *selection of chapters and sections* by *conditional text formatting*.

Note: The best answer to modularity aspects is certainly structured programming techniques, like being suggested with the popular DITA (open source) system. The drawback is: The change of paradigm is easy to execute for a documentation department, but difficult to learn for author’s which do not come from professional documentation departments. As most of a product’s contributing authors are architects and developers, systems like DITA - despite their excellent properties - are no very suitable.

3.2 How to verify that requirements are met

3.2.1 Formal criteria

A set of rules can be established (→ 3.3.4 “Documentation rules”) to verify correctness of architecture documentation. Those rules are given in 10 “Verifying documentation consistency” on page 10-147.

3.2.2 You Only Know it’s “Good Enough” when your readers tell you so!

You can still write *poor* documentation which fully complies to the Formal criteria, in particular if one does not well work in the sense of Req01: Intuitive reading.

Readers of architecture documentation shall therefore regularly be questioned for improvement suggestions and the perceived usefulness of the documentation, in order to reflect the documentation maintenance investments.

3.3 ezRead Documentation Strategy

3.3.1 Using books

ezRead suggests documentation to be organized in books. A book is made of a collection of chapters that describe several aspects of the book's scope. Books can be large because they are actually meant to contain *everything* relevant to their scope.

Product related documents of a project (e.g. architecture papers) become *chapters* of the book.

Like in real books, the product books have a focus on a particular aspect of the product. The titles of the books have been chosen such, that it should be easy to understand, where a particular topic shall be placed. Find more in 4.1 "Book organization" on page 4-35.

> A marketing representative might discuss a new idea with the customer. The technical aspect of this idea will go to the "Book of Requirements".
>
> A developer, however, might have a new technical solution in order to speed up a database. This idea might find its place in the "Book of Architecture" since it does not raise new questions (= requirements) but presenting new answers (= architecture). Depending on the content of this idea it will either go to System-/Software- or Hardware Architecture.

In general the book strategy shall avoid the collection of content in files that are distributed over directory trees. Books consolidate information by **structure**, **content discipline** and **quality aspects** beyond the typical presentation of (good!) papers that are written in a development phase. ezRead Product Documentation is characterized by three different types

The books of Accompanying Documentation	Entry level documentation to help understand the product from a high level perspective. Good for readers who need to learn about the product, conventions, "Where to find" and "How To" questions.
The books of Product Documentation	Documentation that describes the product in a structured approach. This documentation contains all answers about the product but does not have project related information
Set of documents for Project Documentation	Documentation that describes "how to get to the product" in terms of the classical project documents with respect to processes, time, quality, cost aspects. Typically this documentation needs to relate to product documentation

3.3.2 Cross-Referencing aspects

A major aspect of a book is the idea of *cross referencing* information between chapters and throughout books. Unlike a novel, which is typically read top down, most documentation today is used for the fast retrieval of information, which may be called "reference mode".

"Reference mode" means, that a reader very often has a need-to-know for a particular aspect of a product. That reader already knows some context, but needs to dig into a detail of a special aspect of the product.

The ezRead documentation strategy makes extensive use of cross references (also hyperlink) techniques. A reader should be able to get help on technical topics whenever he faces the "Find more detail" problem or the "Get the big picture" problem. Cross references and hyperlinks help in this situation.

Cross Reference: links another chapter in the *same* book (represented as PDF file).

Hyperlink: links a document or URL which is not part of the product documentation, but a reference required to accomplish the documentation.

Note: A typical example of a *hyperlink* is a link to the *data sheet of a microprocessor*. Such a data sheet is available only as a PDF file, often in SECURED mode so that it cannot be changed, sometimes not even printed.

The data sheet is not part of *generated* product documentation but very often needs to be referenced. That type of reference is called a *hyperlink*. URLs to web pages are also considered hyperlinks.

The definition correlates well with the definition of *cross-references* versus *hyperlinks* in Adobe FrameMaker documents. There cross references are considered to link a FrameMaker document to another FrameMaker document, whereas hyperlinks leave the FrameMaker context and link to other file types like a URL or a PDF file. (→ 7.1.1 “Cross References” on page 7-85)

A reader will not percept the difference between a cross-reference and a hyperlink but when *another* file (PDF) or an URL is opened on clicking the hyperlink. This difference is not important to a reader.

3.3.3 Editorial Team

As shown in Figure 3-1, ezRead requires a small editorial team (typically 1..3 persons) as an important quality factor to excellence. Small the team should be because this minimizes “The author diversity problem”. The editorial team acts as a quality gate to the product documentation. A written paper/document can only join the ezRead documentation if it complies to the Documentation rules and if it is “*blessed*” by the editorial team.

Most times, the editorial team will have to change the incoming document to match the standards and ezRead philosophy. Hence there is a evolutionary scheme behind - the more an incoming document already complies with the ezRead standards, the faster it will be integrated into the official document tree.

The editorial team will drive regular review meetings in order to assess the overall readability of the books.

See also 3.4.2 “Why using a selected editor’s team?” on page 3-30.

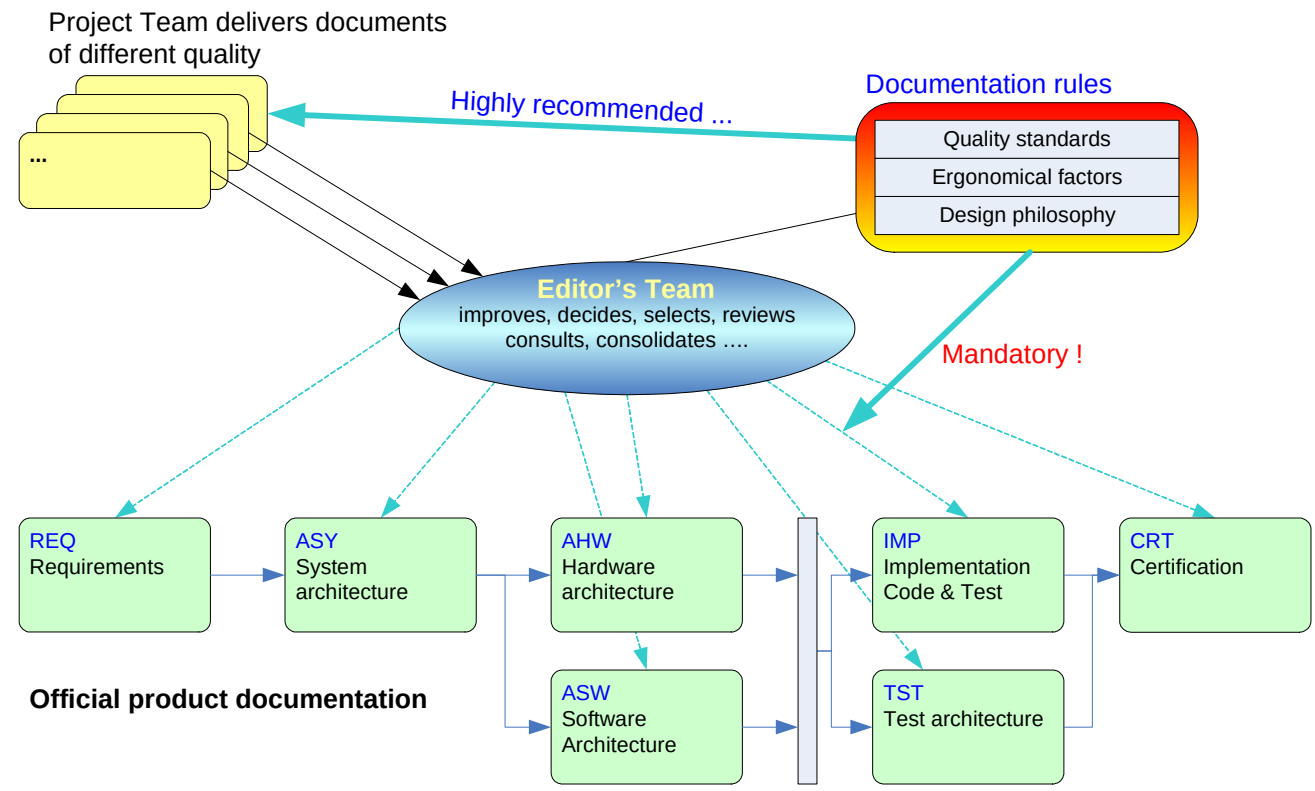


Figure 3-1. Role of the editorial team

3.3.4 Documentation rules

In a team of more than 3 people it starts getting difficult to achieve a common design of documentation. Reasons are discussed in "The directory diversity problem" and "The author diversity problem". Documentation rules help to minimize the corrective efforts of the editorial team. Those rules are published to the entire project group. At best, incoming documentation already complies to the documentation rules. Another helpful approach is the use of Meta Tags Strategy as explained in the next chapter.

The ezRead documentation rules are derived from the well accepted ARC42 template are described in 5 "Writing Architecture documents" on page 5-51.

3.3.5 Meta Tags Strategy

History proves that it is difficult to convince a large team of developers or author's to *create cross reference information*. The reason is, that text is often written in the flow of thoughts and the thin red line of an author's inspiration (yes - even in technical documentation) is sensible to interrupts. Creating cross references is often outspoken as embarrassing because it takes rather long to create cross references - in particular when writing with MS Word.

ezRead has an answer to this problem using the *Meta Tags Strategy.* According to this concept several keywords are available to make cross referencing much easier. Meta Tags Strategy defines tags like *<URS>* indicating a term (here *URS*) to refer to the *glossary.*

A *cross reference* to the chapter of a specification would show up like *<xref ASY:Ch3p5>* and can be written easily without a technical interrupt.

The editorial team has technical tools in order to replace the meta terms by the correct entry. The only condition here is, that a target is specified uniquely. Therefore in the above example, the keyword ASY is available in the cross references *<xref ASY:Ch3.5>* ASY would point to the book of system architecture, the chapter nomenclature is obvious. An extended set of meta terms is available in the coding standards to be used for the same purpose. As code is created with ASCII editors, using meta term strategy is a 'must have' in order to create good documentation.

The actual building process for *ASY:Ch3.5* is not a problem because very often it is that document/chapter anyway, that an author uses right in the moment, that s(he) is authoring text. Unfortunately that precious connection is seldom preserved for the sake of better readability.

It requires less than *five* meta tags to cover more than 95% of the demands that good documentation achieves regarding to such kind of formatting. The relevant tags are described in 6.5 "Using meta-tags" on page 6-72.

3.3.6 PDF export format

ezRead strictly promotes PDF output. PDF is certainly the most popular format worldwide to read documentation and its compatibility is the main driving factor for ezRead's claim.

Another effect is implicitly added when using PDF format. PDF documents cannot be edited but commented[1]. For reviews, the editorial team will provide PDF files that can be commented with the free Adobe Reader. A distributed version of the file is considered *frozen* and comments can be collected from remote sites. Different reviewer's comments are joined into the original PDF document and can then be discussed in a review meeting.

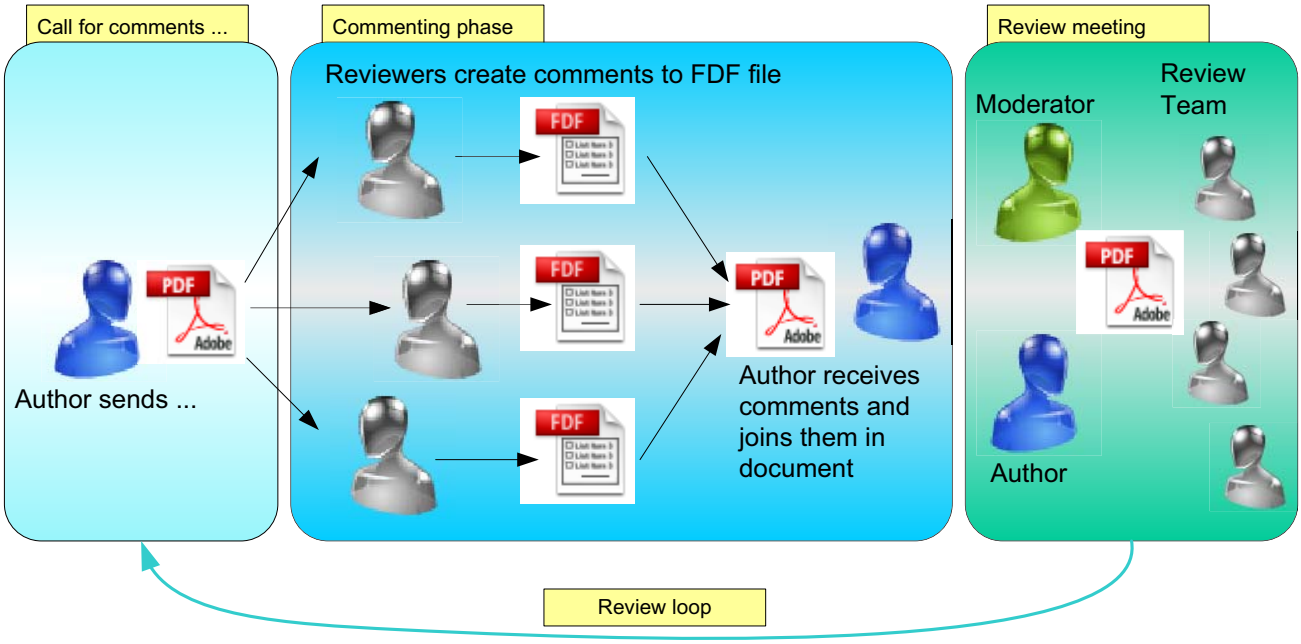


Figure 3-2. Review process using PDF commenting techniques

Note: Although MS Word has developed a commenting technique for documents, this idea has a problem with *parallel* versions. To survive the versioning chaos, feedback can only be given by one person (temporary author) at a time. The PDF approach allows a perfect *parallel processing* which increases the speed of feedback essentially.

1 Of course it is known that PDF files can well be edited, however, this is usable only to patch text rather than allowing a sophisticated editing.

3.4 Questions and answers

3.4.1 Why not a WiKi?

WiKi (abbrev. for Wikipedia) is public internet service with lexical entries for millions of topics. The attractiveness in Wiki comes from the fact, that to a *keyword* there is typically only *one* return entry. This compares to a *search engine*, which may return *thousands of matches* to a keyword.

The *one* entry returning in Wiki actually improves in quality over time because readers may contribute with an update request. An entry owner qualifies the request and might update the entry accordingly.

By this technique the entries' quality increases. All that is a high comfort to a reader. In a sense the Wikipedia idea has a match to the ezRead strategy since both use information consolidation to permanently improve a single point of information.

Industrial departments recently have started their own *project internal Wiki* with good intentions to achieve the same effects like in the internet world.

However, after some time those local small scale Wikis get out-of-sync and are visited less and less until they vegetate as an outdated information grave.

There are mainly two reasons for failing the success of the local Wikis

1. **Wikis are not structured**: Like information in a dictionary isn't structured, a Wiki's entries are not structured. Hence a Wiki cannot be read from top down. Although hyperlinks are possible, Wiki entries must be written like a short article. A chapter in a book can be part of something larger, a Wiki entry hardly can. Being part of something larger, however is a very typical requirement for technical content which often requires a high amount of structure.

2. **Local Wikis lack authoring energy:** The world wide Wikipedia may have 10 Mio. readers of the article "Code review". Only 0.001% = 100 of these readers might be able to contribute to better quality of the article. And only 10% of those 100 readers actually *do* contributions. Finally the Wiki gets the "best of the best" and the article is likely to have high quality content.

   A typical project has a size of 4 to 120 people. The statistical portion of competent contributors might be better than in the world-wide case, but it is still not high enough to get the "quality-momentum" established. Very often there is only one owner per article expressing his view on a topic. There is little variety of opinions to get synthesis effects and hence local Wikis are often more like better diaries of a hand full of authors who believe into Wiki.

   On top of that comes a good portion of project members who do not like neither read nor write Wiki, preferring common tools like MS-Word and EXCEL. Since a Wiki is never the final repository of information, it is often seen as "additional effort" - which *project plan driven* developers are not willing to sacrifice.

3.4.2 Why using a selected editor's team?

Much to that is said before (see Editorial Team). To think that it is possible to train an entire project team with different minds and skill and willingness to generate ezRead documentation is an impossible dream. The energy required to maintain the documentation consistency by controlling the author's will be dimensions higher and more frustrating than a moderate approach with an editorial team and Meta Tags Strategy.

3.4.3 Why asking for high quality documentation at all?

Isn't it good enough what we do and have?

From the point of product documentation, very often the answer is "NO, it isn't" ! The single pieces are ok, the collection of them is not.

Modern products require *reuse concepts*, *modularization*, *inter product interfaces* and *encapsulation*. And while those terms always perform very well on *PowerPoint*, they tend to lack operative execution. But it is exactly the value of high quality documentation which is required to allow for all these features.

Starting early in a project adds only little cost for high quality documentation. It pays off already, when new team members enter the project and have short cycles of work-in time, in particular they do not need asking experts for help.

Maintenance cost are the most critical path that can be reduced by documentation ecellence.

3.4.4 Why don't we have such documentation today?

Most internal product documentation does not meet the few standards set in ezRead. This is, however, little to the point that developers or author's wouldn't be willing to contribute at the standards given here. The main factors come from ...

The author diversity problem: Diversity of author's results in different documentation. ezRead demands an organizational setup in order to achieve consistency. As that is valid for any good documentation, it is often not recognized when a project starts. Document templates only help in a formal way but of cannot control the content, writing style and semantics.

The directory diversity problem **:** Very often in a new project, a directory tree is established. As time goes by, documents are spread all over the directory tree, each of them having a good reason why to exist just there. Often, parts of documentation reside on locally shared drives whereas other parts are stored in version control systems.

The above situation creates a storage diversity which significantly contradicts the requirement of "Find the right entry point fast".

The awareness&planning problem **:** It requires an Editorial Team (1..3 persons) to care for the standards or recommendations. Such a team is today not typical for a project plan. An early start with ezRead minimizes the extra work. After a 2-week startup phase, the work load of the editor's team reduces to about 5 - 20%, depending on the project demands.

The skill problem: Although there is good-will, the *skill* and *paradigm* of ezRead documentation (e.g. cross-referencing) is not available for potential author's. Although that could be easily fixed (best... when a project is started) - it must be fixed and cross barriers when it comes to personal conviction. As a matter of fact from experience, it is impossible to achieve this skill level in large and diverse groups (see "Why using a selected editor's team?")

The promotion problem: Unless at least one *promoter* of ezRead Technology is found, it simply won't start. Whereas ezRead is not a religious question but a cost-saving decision *the fact of cost saving is typically only recognized* when additional (often unplanned) cost arises from the *lack* of readable product documentation.

And that is always late in the project when complexity has grown beyond the point of what a human brain can understand without high quality documentation. Sometimes it even strikes earliest when a project migrates to a *maintenance phase* which is often covered by another than the development team.

***The tools problem*:** Unfortunately the most popular text editor, *MicroSoft Word*, is everything but suitable for the idea of ezRead books and cross references. Professional teams worldwide avoid MicroSoft Word for documentation. *Adobe FrameMaker* has been proven to be a good answer to sophisticated documentation demands and the activities from OpenOffice.org are slowly approaching the Adobe FrameMaker standards. FrameMaker today is unbeaten regarding its ideas of BOOKS which is one of the basic axioms of the ezRead philosophy.

The mix-it-up problem: An frequently found problem is the *lack of separation* between *product*- and *project* documentation. *Project* documentation cannot be organized in books because it uses very different formats like EXCEL, MS Project project plans, DOORS requirement documents.

There is a natural impact of *product* documentation into *project* information. Technical information is required already in the first documents created in a project (e.g. User Requirements specification and a project charter). So documentation starts “from project” right from the beginning.

The demand for organized product and architecture documentation often comes (too) late when - due to project requirements - lots of documents have already been created as a mix of project and product information.

Very often, a functional specification is discussed in the commitment to a customer. The probability is high, if such a product document is referred in contracts. that i*t will contain highly relevant project information*. And the latter spoils the readability as *product* documentation - where a functional specification actually belongs to.

Such a contract-used functional specification cannot contain cross-references because it has to exist as a single unit in the role of the contract reference. If such mix-up is not solved, the basic requirement of readability (Req01: Intuitive reading) cannot be met.

The frustrated author problem: It has often been observed, that good technical people do have a problem to anticipate the “The poor reader’s ignorance” when writing documents, i.e. they cannot understand, that a reader might be hooked off already by not knowing the meaning of an abbreviation.

> *Using XPRssT Technology is a good example. This excellent technology uses XBR elements and OOP extensions on the base of J2EE trustlets. The Bebop Design rules offer a whole new dimension of NNS authoring...*

Reading the text above will frustrate a willing reader and very often (s)he will drop her/his entire attention a few lines later[2]. Excellent technical personal is often frustrated on the other side, to explain again and again to a reader what *XPRssT* means.

Of course the above example can be easily solved using a hyperlink into a glossary (preferred) or an URL on the internet. And of course a less aggressive writing style would do it some other good.

Note: The “frustrated author problem” has been identified as one of the most *significant hurdles* to writing understandable documentation. Again it is not bad will of author’s but the lack of phantasy of what a poor reader might require to understand the technical information.

The poor reader’s ignorance: Given, documentation is written to increase the knowledge of the reader (unlike a legal document which is written to establish legal evidence) the situation is always, that the *writer knows everything* and the reader *does not know everything*.

2 This is by the way the generic problem in bad code reviews

For the writers it is often difficult to understand what it means, that the reader's brain has to form *new connections*, to select, filter, sort, re-organize, understand and drop wrong assumptions. Although our brain is trained to sort of learning, *complex technical content* will put essential stress on the reader due to the nature *that it is complex technical content*.

This complicated brain process is often misunderstood by authors as the "reader's ignorance" but it is everything but ignorance because the opposite is the case. The reader, actively reading a document, does execute complicated brain work. The resources of the brain are a sensitive control system. If the brain is not able to create connections after a certain amount of "loose ends" it will *defend* against further information.

The brain's *defense* acts typically at that point when a reader becomes tired, unconcentrated and even angry (following frustration). And ironically - as a self fulfilling prophecy - by creating bad documentation the poor reader is actually guided into "self-defense ignorance".

One of the things that can avoid this trap is .... *good* documentation!

4 Reading ezRead Documents

4.1 Book organization 35
4.2 Architecture documents 36
4.2.1 Variants 36
4.2.2 REQ - The book of requirements 36
4.2.3 ASY - System architecture 38
4.2.4 ASW - Software architecture 39
4.2.5 AHW - Hardware architecture 40
4.2.6 IMP - Implementation documentation 40
4.2.7 TST - Test architecture 43
4.2.8 CRT - Certification documents 43
4.3 Accompanying documentation 44
4.3.1 Storyboard 44
4.3.2 Documentation Guide 44
4.3.3 Coding conventions 44
4.4 Reference documentation 45
4.5 Project documentation 45
4.6 Tips and hints for reading ezRead 47
4.6.1 Notations in ezRead documentation 47
4.6.2 Working with Adobe Reader 47
4.6.3 How to navigate with Adobe Reader 47

4.1 Book organization

ezRead documentation describes a product in seven books. The number is not mandatory, but a reasonable recommended value. Some of the books only apply under certain conditions (→ "Variants"). Figure 4-1 shows the books of *Product Documentation* in relation to other documentation.

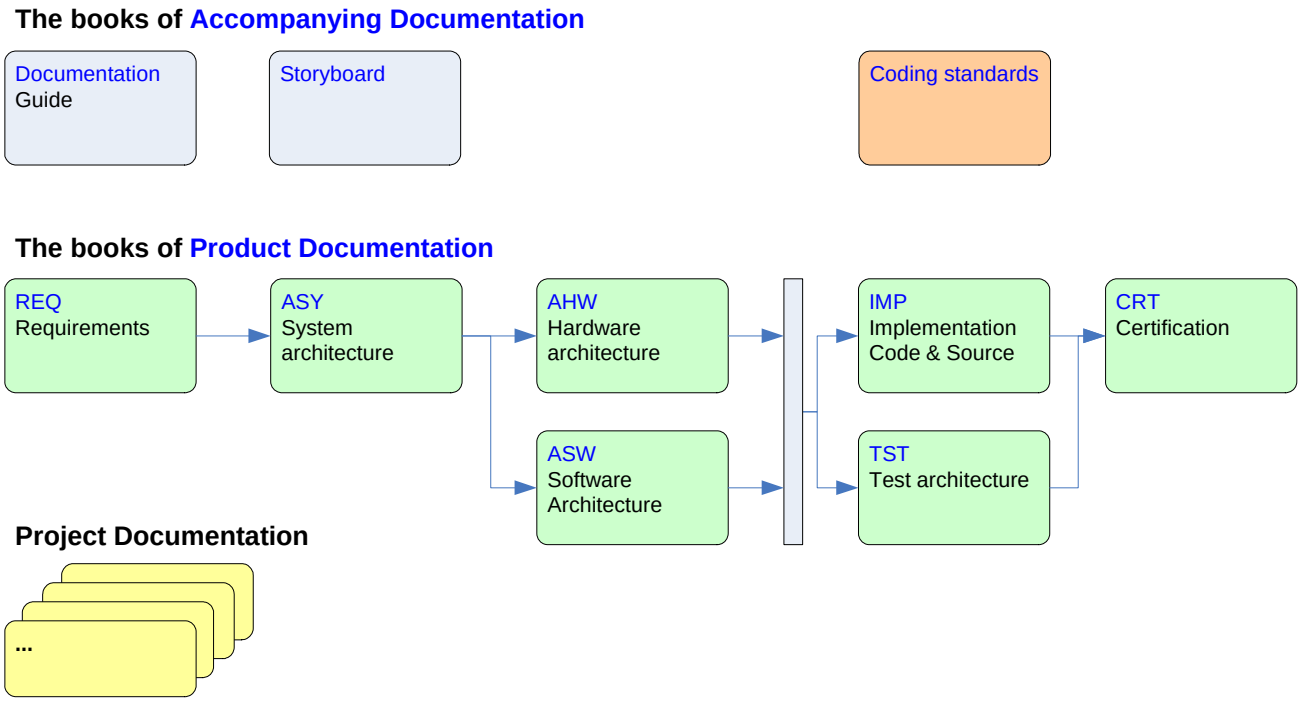


Figure 4-1. Documentation architecture

The different documents are described in the next chapters.

4.2 Architecture documents

The architecture documents comprise the following *recommended* books.

REQ The book of requirements
ASY System Architecture
ASW Software Architecture
AHW Hardware Architecture
IMP Implementation documentation (may be very large and split into sub-volumes)
TST Test documentation
CRT Certification documentation

The proposed abbreviations are suggested to be used throughout the documentation. They are also relevant for using 3.3.5 “Meta Tags Strategy” on page 3-28.

4.2.1 Variants

Some variants should be considered

- In *small projects*, REQ and ASY may be joined in one book.
- AHW is only required if *hardware* is part of the product description.
- The book of certification (CRT) only applies if the product is undergoing a certification according to EAL, FIPS levels or subject to type approvals.

4.2.2 REQ - The book of requirements

Most products are developed and maintained based on requirements. Very often a requirement management system is available to specify the requirements in very high detail. Documents produced under such systems should be referred to by product documentation but they are not ment to be part of it since they are optimized for completeness rather than readability/understandability.

The book of requirements (REQ), on the other hand consolidates the *basic* requirements that lead to the architecture books, described in the following chapters. The focus of the REQ if to provide a model of requirements that allow the architectural layout. The discussion of requirements comprise

- Use cases
- UML diagrams (typ. UseCase and Activity/Sequence diagrams)
- Required standards
- External specifications
- Certification aspects
- Non-functional aspects (Footprint & performance)
- Life cycle aspects
- Features and product variants
- External interfaces
- Architectural axioms
- Compatibility aspects

From the set of those requirements, *architecture* is build as a technical answer to the demands that yield from the REQ.

Statements in the REQ are allowed to refer to architecture, however the opposite direction is more likely because requirements typically do not imply architecture, but architecture answers to requirements.

Figure 2-1 shows how requirements are derived once from requirements management systems and from other external sources like standards or industrial facts. Those additional aspects do not impact the list of system requirements, but they do certainly have an effect on the architecture as described in ASY - System architecture.

4.2.2.1 Working with requirements management systems (e.g. DOORS)

Figure 2-1 gives a rich picture of documentation around a project and a related product. All the books described here are found there whereas in the following we develop a filtered view on the requirements management systems from the perspective of product documentation.

Requirements management systems may well copy from REQ. On top of that, a requirement management system (e.g. DOORS) will derive many other requirements as a result of architecture.

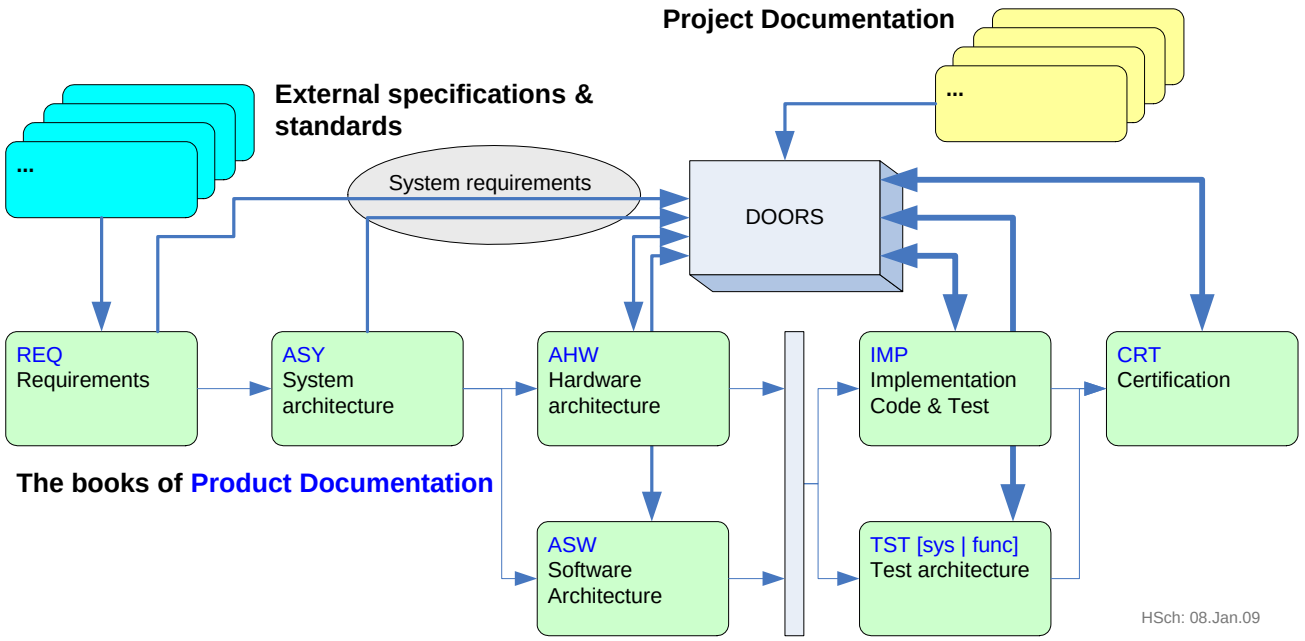


Figure 4-2. The book of requirements and DOORS requirements management

Figure 4-2 shows the difference between requirements in REQ and DOORS. Whereas the DOORS system acquires any detailed requirement, it is typically *difficult to read* for the sake of a complete description of all available requirements.

The REQ book on the other side, *intentionally filters those requirements that are relevant to architectural decisions*. It also provides a more readable layout and possible discussions and rationales why a certain requirement might have been set in place.

Reading REQ versus DOORS is cooperative. Of course requirements from REQ may well support the DOORS system. By nature of their mission, the requirements in REQ are less likely to undergo significant changes than the more detailed requirements in the DOORS system. This is important because of the risk of architectural changes due to requirement changes in REQ can have a drastic impact on development cost.

The latter risk underlines the importance of the REQ and the demand for a high readability which goes hand in hand with a maximum of understandability. The requirements described in REQ shall be well cross-linked into its input documentation to allow a doubtless set of basic claims.

4.2.2.2 Methodology

Written text is mandatory to explain the context of the requirements. UML use case, activity and sequence diagrams should be used in order to support the description of requirements. Also VISIO should be used to make complicated aspects understandable. (“A picture is worth more than thousand words”).

4.2.3 ASY - System architecture

Very often, *system architecture* is badly distinguished from *software architecture*. System architecture knows the customer’s business processes, interfaces, standards and protocols, a knowledge that often exceeds the actually specified system requirements. System architecture specifies functional building blocks, data models, data flow analysis, external interfaces and standards as an answer to the requirements in REQ.

Software architecture is concerned about the optimal translation of system requirements into software technology. It delivers modular software packages, encapsulation, class definitions, library architecture and internal interfaces with respect to maintenance and reuseability.

4.2.3.1 Defining system APIs

System architecture finally defines a set of *System API functions* that allow the elementary communication flow of the system units. That System API might not be literally implemented because the Software architecture may have its own ideas how to realize it. Nevertheless, from a *functional point of view* such an API generated by ASY shall finally be in place.

> **Example:** System architecture might define the functional API to a database like *OpenDB()*. Software architecture could however convert this function call into an actual API definition of *RequestDB()* maintaining the same functionality, but tailored to actual database standards.

If software architecture demands changes in such a system API, then it is possible to put changes in ASY as a result from discussions in ASW as long as that change doesn’t cause significant consequences to other software architecture being in place.

The result of system architecture shall deliver a specification of the entire system such that the basic requirements of REQ are fully addressed. Hence the software architecture does not need to refer to the REQ intensively whereas ASY is very much obliged to those references as being responsible for answering the REQ claims.

4.2.3.2 Propagated requirements from REQ to ASW

It may, however happen, that some requirements of REQ are propagated into ASW and AHW.depending on the nature of the requirement. If, for instance the REQ claims to use an external software library (very often: *certified crypto library*), ASY will propagate this library to ASW as *internal interface*. This is necessary because the suggested (here: crypto) library does not actually show up in the external interface, but indeed represents a pre-defined internal interface, which belongs under the scope of software architecture.

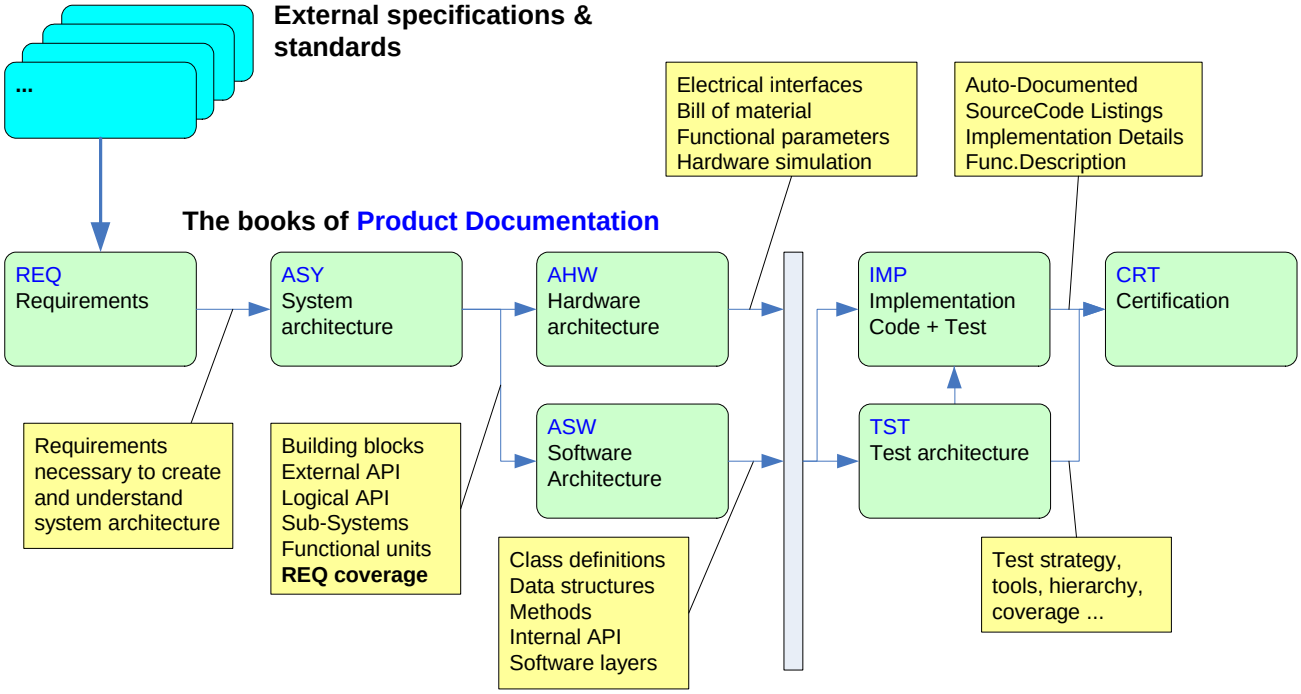


Figure 4-3. The mission of system architecture

4.2.3.3 Methodology

Typically system architecture makes intensive use of UML diagrams. Enterprise Architect is the recommended tool. Find more about recommended tools in “The idea of Meta Tags Strategy is explained in 3.3.5 “Meta Tags Strategy” on page 3-28.” on page 6-72.

4.2.4 ASW - Software architecture

A software architecture takes up the external API functions of the system architecture and optimizes the underlaying functions and modules into a set of programmable modules. In particular *(abstract) classes* will be designed in ASW. The following list comprises the most important definitions made in software architecture

- Modular organization of functional units
- Hierarchy of software components
- Classes and Methods
- Abstract classes
- *Internal* APIs
- Sequence and activity diagrams of internal modules
- Realization of (sub-)systems in the ASY
- Communication flow
- Parallel threads considerations
- Resource allocation
- Access control realization

Definitions should be made to the extend that functional units are described by interface definitions. Any further detailed description can be subject of the implementation which should not be anticipated, but organized by software architecture. The definition of local variables, for instance is out of the scope of software architecture.

Software architecture ASW shall answer to the claims made in system architecture ASY. Consequently it will contain a lot of cross-references into the ASY and vice versa allowing a reader to choose between more detail (from ASY to ASW) or a more distant view (ASW to ASY).

ASW also answers DOORS requirement challenges. Where necessary, the layout of a functional unit shall be supported by UML diagrams or VISIO drawings.

In ASW a reader should find everything to understand, how the architectural proposals and interfaces from the ASY are realized.

4.2.5 AHW - Hardware architecture

Much like the ASY, the AHW describes the functional units to be implemented. For hardware the following aspects shall be covered, if applicable

- Specific design of hardware function blocks
- Circuit diagram/schematics
- Bill of material
- Availability assumptions and replacement alternatives
- Specification of components (e.g. a battery)
- References into external hardware specifications
- Simulation results (if available, can be part of implementation docu)
- Environmental
- Power consumption
- Energy specification and computations
- Performance analysis
- EMC analysis
- Compliance to emission regulations
- Safety aspects

By the nature of the definitions which are to be made in hardware architecture, lots of *cross-references* will typically have to be made into *external specifications* and *regulation papers*. For example a microprocessor's data sheet will have to be referenced in order to make the hardware design understandable. The Editorial Team will provide anchors into the external specifications as described in 6.6 "Preparing books for the final release" on page 6-80.

4.2.5.1 Relation between ASY and ASW

Very often, the hardware design will have a relation to the software architecture ASY. For instance, the availability of a DMA controller will probably impact a corresponding software architecture. The hardware design is likely to have impact on the software implementation. Therefore the hardware architecture AHW should be finalized first.

Reading the hardware architecture AHW a reader shall be able to understand the hardware environment and functional hardware units of the product.

4.2.5.2 Methodology

UML activity diagrams. Also VISIO are should be used to make complicated aspects understandable. ("A picture is worth more than thousand words").

4.2.6 IMP - Implementation documentation

Implementation documentation describes the actual implementation comprising

- Source code listings
- Test case listings
- Additional descriptions

4.2.6.1 Using auto-generated documents

IMP is likely to be automatically generated from auto-commenting tools. This is also the recommended process.

In a *waterfall* approach, implementation can start when software architecture is finalized. In *iterative* approaches, parts of the implementation might even be available as proof-of-concept. Regardless from the software development process it is a good idea to *generate implementation documentation right from the source code* for the sake of a high synchronicity between software architecture and the actual implementation.

Several *auto-documentation tools* are available on the market, most of them as free source code products.

Table 4-1. Market available auto-documentation tools

Product	Description	Pro's	Con's
AUTODOC see "AUTODOC Overview"	Initial AUTODUCK version 2.01 by Erich Arzt (former MicroSoft), became freeware more than 10 years ago. (no GPL license) Improved by Helmut Scherzer (IBM) during 1998 - 2006 adding FrameMaker compatibility, graphics, figures, equations and formatting features. Successfully tested in the IBM "Caernarvon" project (IBM Watson Research activity) AUTODOC is the only tool that works straight into the documentation pool including sophisticated features like graphics, cross references.	• Good description available • Full Adobe FrameMaker compatibility • Tags can be converted to any other formatter • Designed to work for any ASCII like file type. • Special editorial feature • Additional evaluation tools available • All source code freely available and in-house • Function and documentation can be tailored to internal requirements • can be converted from/in other formats	• Not a standard tool in the "auto-documentation" community
JAVADOC	Freeware documentation tool, specialized for JAVA needs.	• popular • in-house skill • Good external documentation available	• only for JAVA code • interprets source code • no FrameMaker support • not as powerful as AUTODOC • field layout is ambiguous
doxygen	Most popular auto-documentation tool, somewhat difficult to learn. Needs configuration effort in order to work for documentation doxygen has a good reputation and a large community. GPL licensed	• most popular tool on the market • Open Source project, source is available • many users, good discussion on Internet	• interprets source code • complicated to start with • not as powerful as AUTODOC • no FrameMaker support • field layout is ambiguous • changes must be made public under GPL lic.

AUTODOC Auto-documentation is described in 2 "AUTODOC Overview" on page 2-51.

4.2.6.2 Adding additional descriptions

Implementation might need additional explanation. For instance, *a special stack mechanism* to optimize task switching might not be under the scope of software architecture, but proposed by the implementation. As a matter of fact, many such situations are likely to occur.

Unless a topic is relevant to be described in the software architecture, an additional chapter shall be added to the IMP to explain the implementation technique. The actual implementation should then have cross-references into that chapter.

Another example is the implementation of a complicated algorithm e.g. a SHA-256 algorithm. Although the ASW may describe such an algorithm simply by an architectural block because such an algorithm is state-of-the art technology, the actual implementation will be complicated to understand. In this case IMP will have to provide a description how such algorithm is implemented and this description goes beyond what is possible by auto-documentation. However auto-documentation should refer to the chapters of such an additional chapter and vice versa.

4.2.7 TST - Test architecture

The test architecture comprises

- Strategy (Unattended, Script based..)
- Tools
- Test Granularity (Unit-/Function-/System-/Integration tests)
- Hierarchy
- Test Coverage analysis
- Version control
- Life cycle considerations
- Test specifications

4.2.7.1 When should TST be created?

Although the test architecture TST does start when ASY is finished, Figure 4-3 implies that TST comes after ASW and AHW. To some extend this is true, test architecture should have some confidence about the product layout, otherwise the test design is difficult to create.

However TST may well be started earlier than the finalization of ASW or AHW respectively.

For instance the *test tools question* is often raised in the early stage of a new product development since appropriate *tools are expensive* and often there is also an availability issue. (e.g. a new processor was launched and the emulator technology is not yet available/trustable).

4.2.8 CRT - Certification documents

Certification documentation follows rules according to the certification scheme (FIPS, CC EALn) in place. Different flavours are possible depending on the target to be met.

- Common Criteria Evaluation
- Type approval

On commont criteria evaluationt typically the CRT contains the

- protection profile
- security target
- TOE
- Detailed chapters describing the evaluation topics demanded by the security target.
- Test reports
- Analysis reports
- References into product documentation which describes aspects of the security target

Publication

Certification documents are created or managed by the Certification Group according to the rules of the designated certification body. Evaluation documents will not be delivered since they may reveal industrial secrets, however the actual certification report is typically published on the website of the certification body.

International mutal recognition is established between certification bodies, e.g. BSI/Germany and DCSSI/France. The extend of mutual recognition can be restricted to a maximum EAL level.

CRT documentation is likely to have many references into the architecture as well as into the implementation documents to prove the proper installation of security.

4.3 Accompanying documentation

Three additional documents are part of the set, which help understand the environment and working conditions under which the product was created.

4.3.1 Storyboard

This book is typically written at the very beginning of product development. It contains chapters about most difficult aspects of the product and its environment, sometimes even of historical quality in a sense that it collects some of the product evolution data.

As the actual product books would not reflect such historical aspect, for the storyboard SBD it is well accepted as feature which can make architectural decisions easier.

The following is a loose collection of items allowed in a

- 30.000 ft. perspective of the project
- Product motivation
- Partner description
- Market
- Competitive analysis
- Tutorial
- Discussion on political landscape for the product
- Team structure

As such the storyboard is a helpful tool for people who enter the project in a later stage and can be used to train a newcomer.

The storybook can also be a contain topics that are not a part of the product, but worthwhile not to get lost. Those topics may comprise creative ideas as well as solutions which had been dropped. Very often - as time goes by -, such information gets lost (also the don’t do’s) but would be important in order not to repeat a bad decision.

4.3.2 Documentation Guide

The chapter you just read is part of a Documentation Guide. The Documentation Guide describes the documentation setup, gives rationale and orientation to users. Other chapters define the documentation conventions, usage of tools and editor’s tips.

4.3.3 Coding conventions

Although today, *code formatters* are freely available, *coding conventions* are still required for aspects beyond indention and the placement of braces. *Naming conventions* and coding style cannot be entirely covered by those tools.

The coding conventions are important to be followed in large projects and it is well known how difficult this is to achieve with different people. If followed, however, cost for evaluation will decrease, quality of code will improve by better understandable code review.

Coding conventions also support the *auto-documentation* aspects. In order to produce hight quality (auto-)documentation programmer's have to develop a particular *commenting style*. If this is followed up at the beginning of coding, the positive effects will pay of soon.

4.4 Reference documentation

Reference documentation is documentation created *outside* the scope of Architecture documents. Very often Project documentation requires to write up essential technical contributions, e.g. as part of a functional specification or a STARS. If those files are available (as PDF), it is ver practical to refer to their content rather doing the effort of cut & paste integration into product documentation.

> By the techniques suggested with ezRead strategy, those reference documents shall be equipped with anchor points, called "Preparing books for the final release", at the granularity of chapters or even sections (→ 7.1.4 "How to create a named destination manually" on page 7-88).

Referring to project documentation mostly applies to the *early phase of a project*. After project finalization, *product documentation* should be entirely independent from *project documentation*. Reference documentation is recommended to be stored in the *...\XY00_RefDocu*. (→ 6.4 "Where to put documents" on page 6-70).

4.5 Project documentation

Little needs to be said about project documentation since much has been said in hundreds of excellent books. Summarizing, *project documentation* is not considered to be part of *product documentation* and does not need to follow its rules and recommendations.

Typically project documentation comprises several of the following documents

- Project Charter
- Project Companion
- Project Plan
- Resource Allocation
- Time - Quality - Cost priorities
- Risk Plan
- Quality Plan
- Status Reports
- Stakeholder Requirement Specification (STARS)
- and others

Those are likely to be available in most *different formats* like EXCEL, MsProject (MPP), DOC, MindMap (MMAP) and more. Whereas the idea of "only PDF" (see 3.3.6 "PDF export format" on page 3-29) as transport format can be maintained for the *product* documentation, it becomes quite unreasonable for *project* documentation.

Project documentation is rather difficult to maintain regarding readability standards because *many* changes have to be made by *many* people with *little time* for caring for readability aspects. Very often project documentation is not read *more than once* (e.g. on reporting) or in *minutes*. There is actually no winning point in applying ezRead methodology to *project* documentation

Therefore the design and layout for *project documentation* is not under the scope of this documentation approach.

4.6 Tips and hints for reading ezRead

If ezRead documentation was meant for *intuitive reading*, it should not require much of an explanation. Some hints, however, shall be given nevertheless

4.6.1 Notations in ezRead documentation

Anchor Words are printed in *italics* to emphasize the focus of a statement. They help a reader to catch the idea of a statement. This is better than **bold** printing which is only used if the importance of a statement shall be heavily emphasized. When carefully used, *anchor words* improve the intuitive readability significantly. Very often these are those words which a reader would emphasize when reading loud.

Special attention is expressed through the Highlight font which appears as a lighter text.

Hyperlinks are typically blue but other than in many internet hyperlinks there is no underlining. This is practical in printed documentation where hyperlinks are not available and hence will not invite to expect information. In documents with intensive hyperlinks it has been succesfully practised to use a more moderate (darker) notation) for highlighting in order to avoid the reader to be distracted by chaotic contrast.

> This is a text note
>
> Sometimes a text note appears. This is typically a comment, tip or an example which can be seen as encapsulated statement. It is often helpful information but not necessary to understand the text.

4.6.2 Working with Adobe Reader

PDF has been chosen as the preferred deployment format (see 3.3.6 “PDF export format” on page 3-29). Most readers have the freely available Adobe Reader, some have the Adobe Acrobat version (at cost) and even fewer are lucky owners of Adobe Acrobat Professional.

Regardless which version you own, reading ezRead documentation is always possible. For use with Adobe Reader (Free version) the editor’s team will apply a special function in the *Professional* version to allow commenting with the free Adobe Reader.

4.6.3 How to navigate with Adobe Reader

Adobe removed some attractive features with the introduction of Acrobat X (and following). One of the removals concerns the navigation to the parent document. Therefore we explain navigation for the versions prior to Acrobat X and later.

- 4.6.3.1 “Navigation prior to Acrobat X”
- 4.6.3.2 “Navigation with Acrobat X and later”

4.6.3.1 Navigation prior to Acrobat X

There is no need to explain using a hyperlink. However, when a hyperlink jumps into child PDF file the question is, *how to navigate back into the parent file*. There are two choices

Using mouse control. Adobe Reader provides *navigation buttons* which can be selected by the mouse. To use these navigations buttons do the following:
Select *View - Toolbars - More Tools*

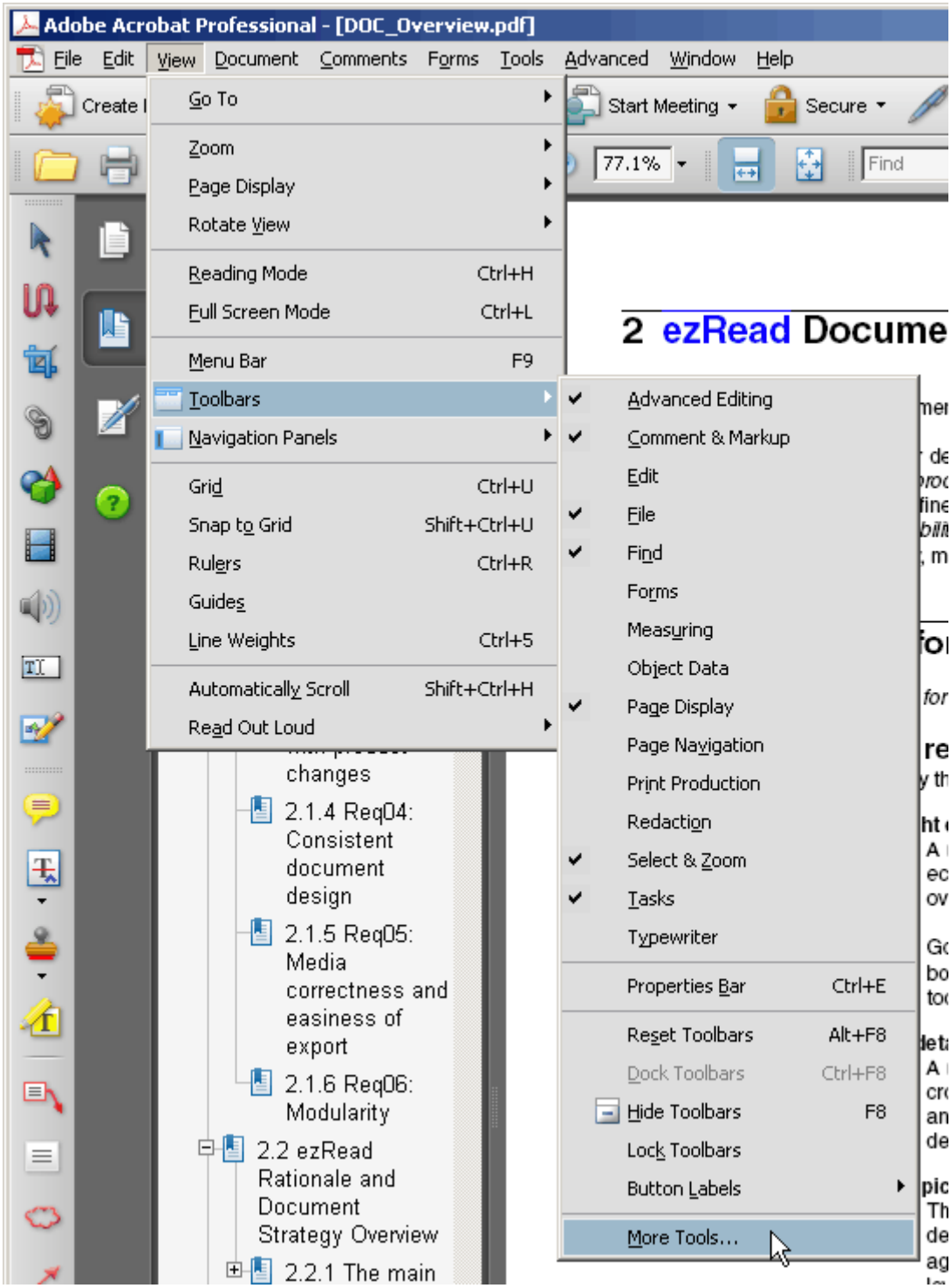


Figure 4-4. Installation of navigation buttons

3. On the next selection panel choose the *Page Navigation Toolbar* settings

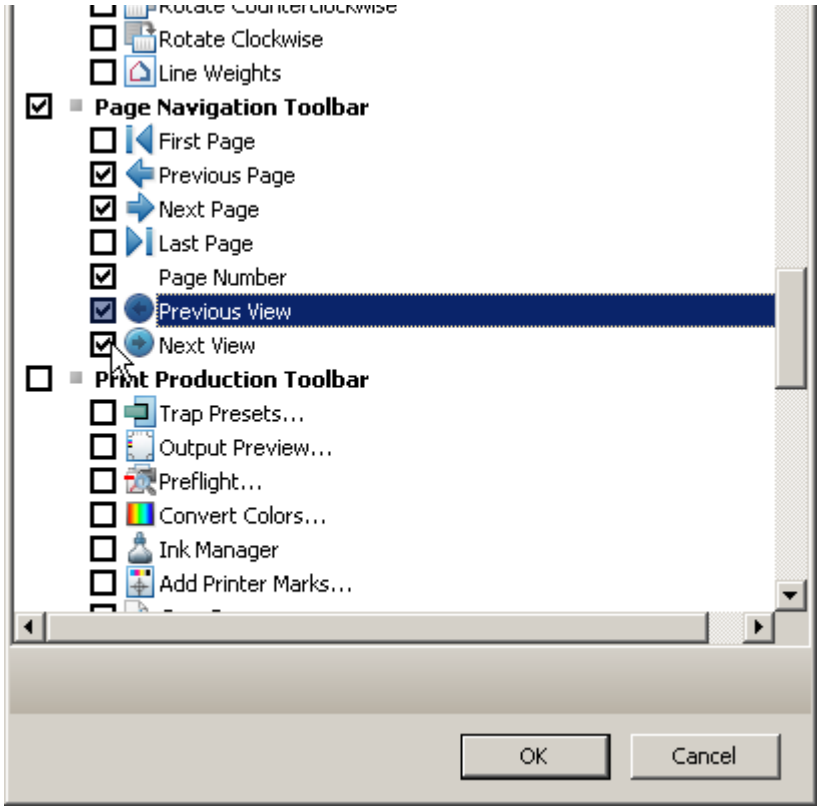


Figure 4-5. Check Previous and Next View

Confirm with “OK”.

4. On the Toolbar, two blue buttons 

appear, which can be used to navigate to the *previous* and *next* location. The *advantage* of using the mouse is the *ease of use*.

   The *disadvantage*, however is, that there is no shortcut to navigate to the *parent* ***document***. If you did *just* hyperlink into *child* PDF file the back arrow key will of course bring you to the parent document.

   However, if you did scroll down several pages in the child document, perhaps jumped to some more pages, then the navigate button would make you retrieve all these steps until you reach the entry that brought you into the child PDF.

   There is no easy way to navigate to the parent PDF unless you rewind to each of your single actions. *Key control* solves this problem.

Using key control: The “r*ewind the stack*” problem, can be solved when using key control Press

ALT + <cursor left>
to navigate to the previous *instance* (this corresponds to the key.

ALT + SHIFT + <cursor left>
to navigate to the *parent document*.

Key control is therefore the recommended method to navigate through documents

4.6.3.2 Navigation with Acrobat X and later

Acrobat X has dropped some valuable functions.

For instance the **ALT+SHIFT + <cursor left>** to move to the *parent document* was removed with Acrobat X. Navigation between linked documents, however, is important for an efficient workflow and consequently this function has now been added to the ND.API.

Using mouse control: Adobe Reader provides *navigation buttons* which can be selected by the mouse. To use these navigations buttons do the following:
Select *View – Show/Hide – Toolbar Items – Page Navigation* – [*Previous*|*Next*] *View*

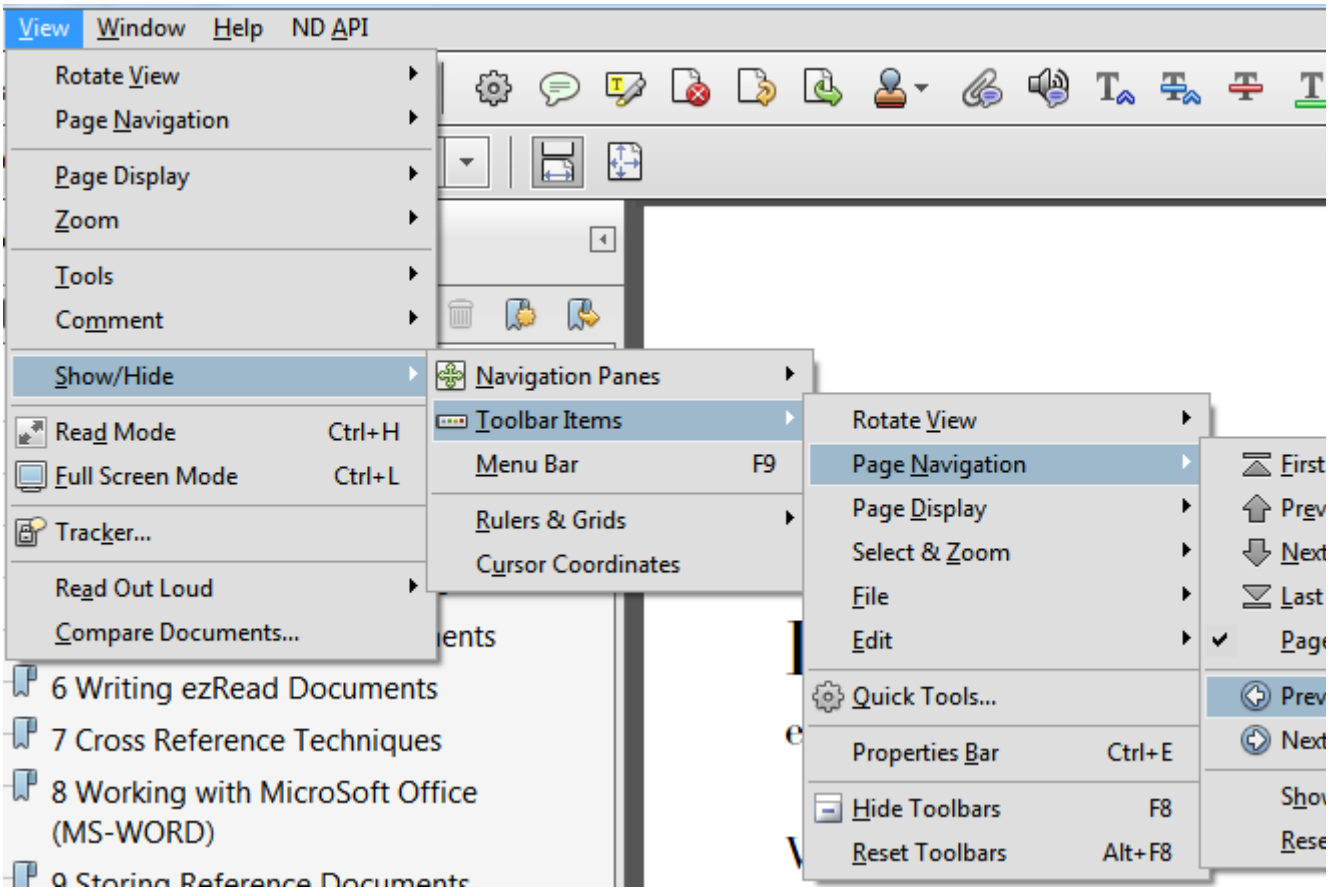


Figure 4-6. Installation of navigation buttons in Acrobat X

These buttons, however, will not allow to jump to a parent document. but they do work within a document. Hence use the functions in the ND.API which are described in 12.1.1 “Navigate Documents”.

Using key control: The use of key control is obvious since the menu items in 12.1.1 “Navigate Documents” can be easily executed by the associated *Alt–<key>* combinations.

ALT+x p: navigates to the parent document

ALT+x c: navigates to the child document (previously selected as link)

5 Writing Architecture documents

5.1 Main Concepts 51
5.1.1 Use a structure template for documentation 52
5.1.2 Architectural views to separate concerns 52
5.1.3 Top-Down approach to manage complexity 52
5.1.4 Blackboxes to describe responsibility and interfaces 53
5.2 Architecture documentation structure overview 54

Writing architecture documentation is only one aspect of Using books in ezRead. However they make the most important contribution to understanding. The low level documentation (IMP and TST) may partly be generated automatically but may well follow the general aspects described in 3.3 “ezRead Documentation Strategy” on page 3-26 and 6.1 “Style recommendations” on page 6-64). For architecture documentation the Documentation rules are given in this chapter.

The documentation rules as described here are fully compatible with the ARC42 specification whereas the ezRead Documentation Strategy adds some general (important) aspects to the ARC42 proposal.

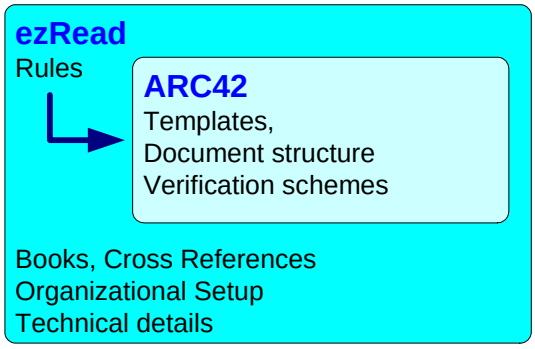


Figure 5-1. ezRead system integrates ARC42

5.1 Main Concepts

Three main concepts shall be followed by writers

- The template of 5.2 “Architecture documentation structure overview” shall be used for architecture documentation.
- All *image sources* shall be delivered with the chapter/document
- Cross references shall be provided. Using meta-tags is allowed.

The Editorial Team will later migrate an architectural document into a book chapter and resolve the meta-tags.

There is a general principle in architecture to be understood – Be pragmatic and efficient !

There is no point to create different views and architectural documentation if the target is easy to understand. What to create for an architecture, should always be a reasonable decision and not follow strict rules. The chapters below, however, want to make the author competent to decide what to apply in order to the final goal – making documents understandable.

5.1.1 Use a structure template for documentation

Architecture documentation relies on a *standardized structure* of its documentation:

- Chapters and sections of the documentation are *predefined* and fixed and in a specific order.
- Each section has a predefined purpose and scope of content (→ 5.2 “Architecture documentation structure overview”)

This structure has been derived from the ARC42 documentation template and is widely used in organizations from various industries.

5.1.2 Architectural views to separate concerns

Any software architecture consists of *several distinct structures*, which have to be documented separately in so called “*architecture views*[3]”.

Think of blueprints in normal building construction – there is one to show the outline of walls, doors and windows, another one to document heat- and water pipes and even another one for electric cables. Such a distinction shall be made in documenting and designing software systems. The following four views are highly recommended for software architecture (→ 5.2 “Architecture documentation structure overview” on page 5-54 for samples):

- **Context views:** Show the system in the context of its *neighbor systems*. You can document the following context information:
  - *Static* (or building block) context, showing “logical” building blocks. Every neighbor system is represented as distinct building block. All channels leading in or out of the system are depicted.
  - *Deployment context*, showing the topmost deployment units or artifacts for the system.
  - *Runtime context*, showing the most important use-cases of the system in context of its neighbors.
- **Building Block views:** Show the *static structure* of the system in terms of building blocks. They answer the question: “How is the system constructed from source code”? Every building block has a corresponding representation in source code.

  Examples: A building block can be a java package or a single java class, a set of (related) classes, a configuration file, a database script or a similar artifact.
- **Runtime views:** Show the *behavior* of the system, namely the runtime-behavior of its building blocks interacting with each other and the neighbor systems. They answer the question: “How does the system actually work, how are system functions being performed?” Runtime views often have a strong correspondence to use-cases.
- **Deployment views:** Show the artifacts used to actually run the system together with the required hard- and software. They answer the question “How and where is my system installed and what are the infrastructure requirements?”

5.1.3 Top-Down approach to manage complexity

In documentation of software systems one shall always begin with a *top-level overview*. Further details shall be added where necessary (top-down refinement). This approach uses two kinds of abstraction-mechanisms:

3 This idea was originally published by Philippe Kruchten, inventor of the Rational Unified Process and other cool things

- *Blackboxes*: These are described by their interfaces and functions only. Inner workings or inner structure is completely hidden – abstracted away. Blackboxes follow the information hiding principle, they hide their inner complexity. For further info on blackboxes see 5.1.4 “Blackboxes to describe responsibility and interfaces”.

- *Whiteboxes*: These are “opened-up” blackboxes, revealing inner structure and inner workings. Whiteboxes themselves consist of a number of (smaller) blackboxes.

Blackboxes can be refined by “opening” them, turning them into whiteboxes. At a certain level of detail, an architect will delegate the inner working of any blackbox to a developer, letting them design the implementation of this specific blackbox.

In a similar fashion, the author of architecture documentation may refine and enhance the documentation for the function of a Blackbox by adding the relevant document for the underlying Whiteboxes.

The architecture documentation shall follow this top-down approach: Beginning with a *context view* as defined under 5.1.2 “Architectural views to separate concerns” (called *level 0*, where the system itself is represented as a black-box) and drilling down, until an appropriate level of detail is reached. Usually three to five such levels are sufficient to describe a software architecture for most stakeholders.

5.1.4 Blackboxes to describe responsibility and interfaces

Blackboxes are architecturally relevant building blocks of the system. They are documented with their purpose and responsibility and their interfaces:

- *Purpose* and *responsibility* briefly describes why this blackbox is relevant for the architecture, its main tasks or reason for its existence.

- According to the IPO-Model[4] (German: “EVA-Prinzip”), a blackbox receives a certain input via its incoming interface, performs its processing and yields certain results via its outgoing interface. Those interfaces have to be described in order to characterize the blackbox.

Blackboxes provide the link between architecture, design and implementation of the system: On the lowest architectural level, the blackboxes either represent source code themselves or shall be linked to a number of source code artifacts.

4 See http://en.wikipedia.org/wiki/IPO_Model for details

5.2 Architecture documentation structure overview

Overview and explanation of the template structure. **N** is the number of the main chapter.

N - Architectural Topic (e.g. “Clustering Concept”)

Describe the idea of the architectural topic in the form of a press release.

N.1 Introduction and Goals

This section shall describe the *fundamental requirements* that software architecture must fulfil. It summarizes *functional* and *non-functional* (quality) requirements.

N.1.1 Business Requirements Overview

Overview of *business* requirements. References to details requirements documentation, only minimal redundancy.

In the ezRead Book approach (→ 3.3.1 “Using books” on page 3-26) this chapter shall list the requirements relevant for the particular chapter in the form of *cross references* to the books of requirements. Using the headline as output format (e.g. “Req03: Consolidated information”) if often good enough to list a requirement without further explanation.

N.1.2 Stakeholder

A list of persons or organizations or organizational units influenced by or influencing the architecture.

N.1.3 Architecture goals

The most important goals for the architecture. Please note these goals will most likely differ from the project goals of any implementation or release project. Quality requirements, like maintainability, flexibility, understandability belong to this category.

N.2 Architecture constraints

Describes all factors limiting design decisions.

N.2.1 Technical constraints

Hardware-, software and operational constraints (like operating systems, application servers, middleware etc.), development constraints (like version management systems, development process constraints etc.)

N.2.2 Organizational constraints

Organizational standards and conventions, *legal* constraints.

Companies often have a set of process documents relevant to the particular project. The use of GPL-licensed programs is a good example for such a situation. A company might want to restrict or allow the use of GPL-licensed programs under particular conditions which have to be respected.

Cross references shall be given in to the appropriate organizational documents

N.2.3 Conventions constraining architecture and development

Programming and *documentation* guidelines, coding- or naming conventions, guidelines for version-, configuration- or test-management.

Cross references shall be given in to the appropriate organizational documents. For instance, references shall be given to this particular book as part of documentation guidelines.

N.3 Context View

As mentioned in 5.1.2, *different architectural views* support understanding of the system as a whole. Chapter 3 to 6 of the architecture documentation are dedicated to these views, beginning with the top-level perspective, the *context view*.

The *context view* section describes the whole system as a *blackbox* with all its external interfaces. All relevant aspects of those interfaces shall be briefly described here:

- Kind of transmitted data
- Format of transmitted data
- Transmission media
- Transmission metadata (frequency of transmission, security, compression, transformation, (a-)synchronicity, monitoring, verification, handshake)

Alternatively, the *details* can be described in the corresponding building block sections.

The context view is refined in the building-block view in chapter 4 of the documentation template by revealing the inner structure of the system, hierarchically refined.

N.3.1 Business (logical) Context

Describes the function and context of the system from a business point of view, e.g. references to the *major use-cases*. Context can be documented using static UML diagrams, business processes or functions by *sequence* or *activity* diagrams.

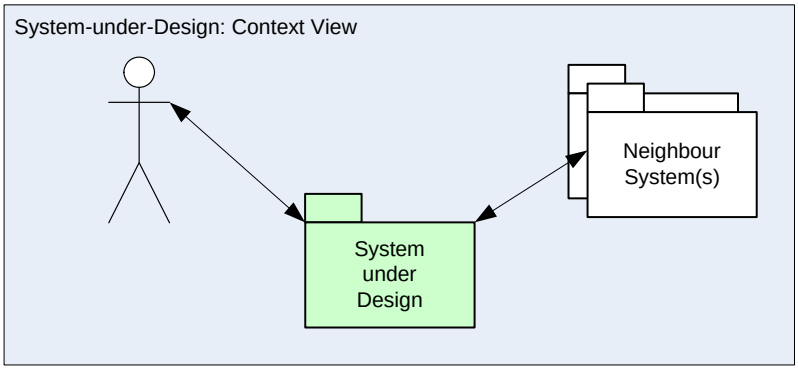


Figure 5-2. Sample of context view

N.3.2 Physical Context

Describe *channels* and *media* between system, its neighbors and its environment. Use *UML deployment diagram* plus textual description of elements.

N.4 Building Block View

The static architectural structure, describing the *building blocks* (subsystems, modules, components etc.) of the system. A building block is a structural unit, which has a representation in *source code artifacts*. This is the *most important view*, usually documented by a combination from static UML diagrams plus textual descriptions.

This structure is the result of architectural creativity, of the software-/system architect deciding upon how to break down the system into building blocks.

The building block description is organized by architectural levels, with the context view from the preceding chapter being level 0. Blackbox and whitebox descriptions are used alternately, as depicted in Figure 5-3.

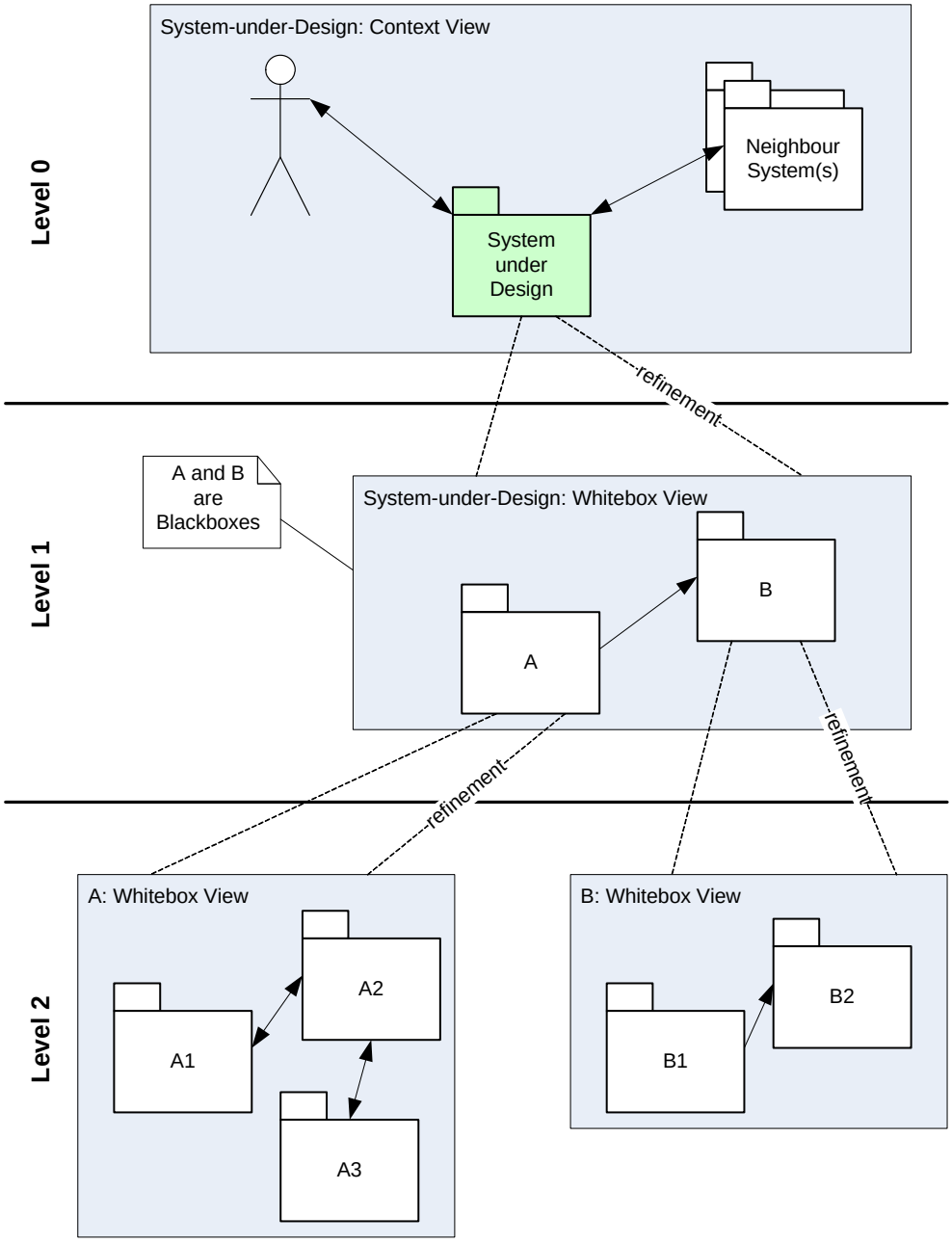


Figure 5-3. Black- and Whiteboxdescriptions in building block hierarchy

In Figure 5-3 “level 1” is the whitebox of “level 0”. On level 2 the two different blackboxes from level 1 (named A and B) are refined in two distinct whitebox views. This kind of refinement is used throughout the building block view: Every blackbox can be refined by a whitebox at the next level of detail. This concept yields a tree structure of building blocks.

Within the documentation, this tree is mapped to the linear section structure:

- Level 1 of this view is documented in section N.4.1 “Building Block View Detail Level 1”, where every building block gets its own subsection N.4.1.1 “Building Block 1 (Blackbox Description)” ff..
- Level 2 is documented in section N.4.2 “Building Block View Detail Level 2” and so forth.

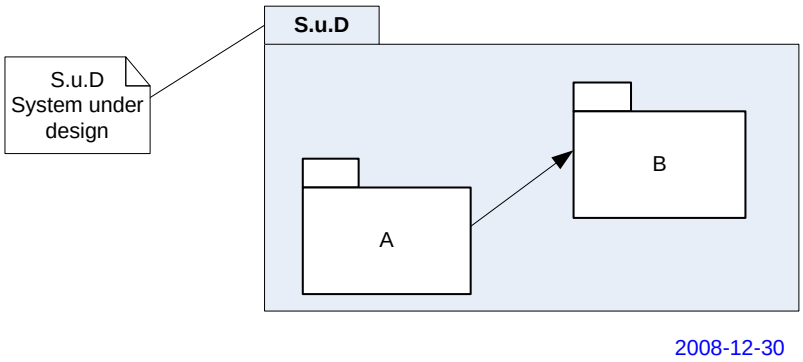
The further one reads within this part of the architecture documentation, the more details are revealed about the system.

N.4.1 Building Block View Detail Level 1

The first part of any refinement shall always be a *whitebox description*, detailing the corresponding blackbox of the previous level. Here, in level 1, this is the refinement of the whole system, its decomposition in layers, subsystems or similar building blocks.

Every whitebox description consists of the following elements:

Table 5-1. Elements of a whitebox

Element	Description
Overview diagram	A diagram (UML packet, class or component diagram) depicting the *structural elements* and their relationships and dependencies. Take this example (excerpt from Figure 2)  *Figure 5-4. Whitebox of System-under-Design*
Local building blocks	Table or list with names *brief descriptions of local building blocks*. These local building blocks are all blackboxes! Their inner structure is revealed in the next level of detail. In the example above this would be two blackboxes A and B.
Local relationships and dependencies	Table or list with brief descriptions of the *dependencies*. In the example, this would be the relationship between A and B.
Design Decisions	Reasons or design decisions leading to this structure.
(optional)	Discarded design alternatives
(optional) References and further information	This section is a placeholder for other kinds of information one needs to understand this particular whitebox.
(optional) Open issues	

Now that one knows the structure of this part of the system its necessary to describe all the blackboxes in more detail, especially their purpose and their interfaces. That shall be done in the following subsections of the documentation conforming to the following scheme:

- Local building block A is detailed as blackbox in section N.4.1.1 “Building Block 1 (Blackbox Description)” (and, with more detail, as a whitebox in section N.4.2.1 “Whitebox view of Building Block A, Level 2”)

- Local building block B is detailed as blackbox in section N.4.1.2 “Building Block 2 (Blackbox Description)” (and, with more detail, as a whitebox in section 4.2.2)

N.4.1.1 Building Block 1 (Blackbox Description)

Every blackbox shall be described according to the following scheme:

Table 5-2. Blackbox descriptors

Element	Description
Purpose & Responsibility	Describe the purpose or responsibility of this building block from the perspective of its users. Keep this description brief, one to three sentences.
Interface(s)	Describe the *inbound* and *outbound* interfaces: what the building block delivers to others (outbound, provided interfaces) and what it expects from others (inbound, required interfaces)
Fulfilled requirements	References or links to *requirements* that this building block either fulfills or supports.
Variability	What may change about this building block or what is *configurable*
Location/File	A mapping of this building block to *source artifacts* (packages, classes, style sheets or any other artifact). In case the building block is “large” or abstract, concentrate on the “important” artifacts: It is often superfluous to enumerate all associated java classes here, just mention the *important* ones.
Further info	Author, Change history, version information and other things not found elsewhere.
Open issues	Open questions, issues or risks associated with this building block.

On detailed levels of building-block descriptions, the *major artifacts needed to implement the building block* shall be referenced in section location/important files. The latter section should contain the main entry point (e.g. main class or classes, sitemap etc.) plus brief info on depencend artifacts (like build files). Additional information is optional.

> Refrain from giving too many details here. Usually developers use their development environment to determine the current and complete set of required artifacts, they do not consult written documentation for that purpose.
>
> Furthermore there are substantial maintenance costs associated with completeness, especially when in keeping a complete list of code artifacts.

N.4.1.2 Building Block 2 (Blackbox Description)

This shall be the blackbox description of the next building block from section N.4.1 “Building Block View Detail Level 1”. In the small example this would be building block “B”.

The structure is exactly the same as shown in Table 5-2, describing purpose & responsibility, interfaces, requirements, variability, location & files etc.

N.4.2 Building Block View Detail Level 2

In the proceeding section the level 1 building blocks have been documented as blackboxes. This current section (4.2) shows the next level of detail. In our example we need two subsections:

- Blackbox “A” from level 1 is detailed as whitebox in section N.4.2.1 “Whitebox view of Building Block A, Level 2”
- Blackbox “B” from level 1 is detailed as whitebox in section N.4.2.2 “Whitebox view of Building Block B, Level 2”

N.4.2.1 Whitebox view of Building Block A, Level 2

In our example this is “A” from level 1. At first we describe this building block as whitebox, revealing its inner structure, together with the description elements already shown under section 4.1 of the template.

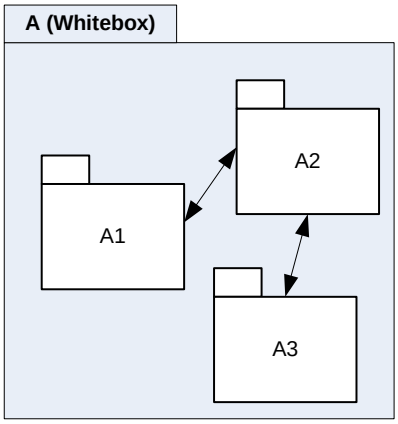


Figure 5-5. Sample Whitebox Overview of Building-Block „A“ (Level 2)

N.4.2.1.1 Building Block A1, Level 2, (Blackbox Description)

Blackbox description of building block A1, again conforming to the structure already mentioned above, describing purpose and responsibilities, interfaces, requirements, variability, location and files etc.

This documentation pattern (alternating white- and blackboxes) continues up to the granularity *appropriate for the target audience* of the documentation:

- *Development stakeholders* need more details, to enable efficient mapping of building blocks to code artifacts.
- *Management*, operations and testing/QA stakeholders need less detail.

In the book approach, it is recommended, to separate sub-architectures into other chapters. The cut is a subjective decision and should be driven by the idea of reasonable packaging.

N.4.2.2 Whitebox view of Building Block B, Level 2

develops correspondingly to N.4.2.1 “Whitebox view of Building Block A, Level 2”

N.4.3 Building Block View Detail Level 3

develops as above 4.1 and 4.2.

N.4.4 Generic dependencies & patterns

The number of this section depends on the depth of the building-block tree (see Figure 5-3). With three levels of details number of this section is 4.4, but could also be any higher *Head2* numbering.

Sometimes the *pure hierarchical* decomposition in the different architectural levels does not suffice to describe all necessary dependencies between architecture building blocks. In that case you might document either g*eneric or specific dependencies* or *patterns* here:

- *Generic dependencies* or *patterns* occur in multiple places within the architecture.
- *Specific dependencies* occur only once and combine building blocks from different levels or places into one view.

Any number of such dependencies can be documented here, each one in its own subsection.

N.4.4.1 Generic dependency 1

One specific example of a generic dependency or pattern is the *Cocoon-way* of transformation pipelines. (→ Wikipedia "Apache-Cocoon")

N.5 Runtime View

This view describes how the systems' building blocks *interact at runtime* in *different scenarios*:

- How are the most important *use-cases* handled?
- How to (instances of) building blocks interact with their *environment*
- How is the system *bootstrapped* and/or started?
- How are *exceptions*, system failures and error conditions handled?
- How are *external interfaces* with their protocols (import/export to external systems) handled?

Use any *dynamic UML diagram* (e.g. *activity*-, *sequence*-, *object*-diagram).

N.5.1 Runtime Scenario 1

Every scenario is documented with a diagram plus (optional) additional text within its own subsection.

N.6 Deployment View

Describes the various *execution environments* for the system, especially processors (nodes, where software building blocks are executed) and channels (over which data is transmitted).

We distinguish between production, test and development environment, which can be described in their own subsections.

The respective stakeholders shall use this part of the documentation to properly install (and configure) the system in their environment.

Use *UML deployment diagrams* to depict nodes and channels.

Note: Nodes can be nested. Every deployment- artifact shall be described here (e.g. every file)

The structure of this view can vary:

- Every environment in a subsection
- Every node or type-of-node in a subsection

N.6.1 Production environment

N.6.1.1 Production Node 1

Describe the node with its required features. Most important here is the *association with building blocks*: Building blocks will be executed on one or several nodes – this mapping shall be described in this view.

N.7 Design Decisions

Document major design decisions, at least those with either:

- long-lasting impact
- chances of surprising future developers or architects
- high risk
- high impact on any quality attribute

These decisions are documented in plain text without formal constraints.

N.8 Architecture Aspects

Architecture aspects are all crosscutting concerns which are not handled by a single building block within the application architecture or which are common to a number of building blocks.

N.9 INDEX

The entries of the chapter shall be marked for automatic indexing to allow topics easy to find. How to generate an index can be found in FrameMaker Guide Chapter 17: Hypertext and view-only documents and Chapter 10. “Creating a Master Index” in volume 2 page 10-1.

For writers not using FrameMaker, please use the meta-tag <ix *marker*> tag to help the editorial team for the architecture documentation.

Maintain an index of all major and important topics within the architecture documentation. The following topics shall be included in the index:

- The *names of all building blocks*, relationships, associations, nodes and channels
- The names of *major code artifacts*.
- All *architectural aspects*, keywords and patterns.
- All major *nonfunctional requirements* (their definition and decisions concerning their solution)

6 Writing ezRead Documents

6.1 Style recommendations . . . 64
6.1.1 Using standardized paragraph styles . . . 64
6.1.2 Using highlighting techniques . . . 64
6.1.3 Using hyperlinks (Cross-references) . . . 64
6.2 Using Graphics . . . 66
6.2.1 Include time stamps to architectural images . . . 66
6.2.2 Using VISIO graphics . . . 66
6.2.3 Using Image import . . . 66
6.3 Using formatting templates . . . 67
6.3.1 The FormatRef.FM file . . . 67
6.3.2 The SysVars.FM file . . . 69
6.3.3 Don't care for document layouts . . . 69
6.4 Where to put documents . . . 70
6.5 Using meta-tags . . . 72
6.6 Preparing books for the final release . . . 80
6.7 Version management . . . 81
6.7.1 Static targets . . . 81
6.7.2 Dynamic targets . . . 81
6.7.3 Target tagging techniques . . . 81
6.7.4 Chapter number oriented referencing . . . 82
6.7.5 Named Tag oriented referencing . . . 82
6.7.6 FrameMaker referencing . . . 82
6.7.7 Best practices to overcome the 'moving target'-Problem . . . 82
6.7.8 Content oriented version management . . . 83

This chapter contains information that an editor should know and which are *mandatory* for the Editorial Team. Whereas the previous chapter "Writing Architecture documents" made a focus on the general structure of architecture documentation, this chapter adds important stylistic aspects of the ezRead philosophy (→ Figure 5-1).

The purpose of this chapter is to invite *any author*, writing technical documentation, to follow the guidelines given here. Whereas the suggested practise of this chapter is "*highly recommended*" (see Figure 3-1), the editorial team *guarantees* the operation in the books of *product* documentation and *accompanying* documentation.

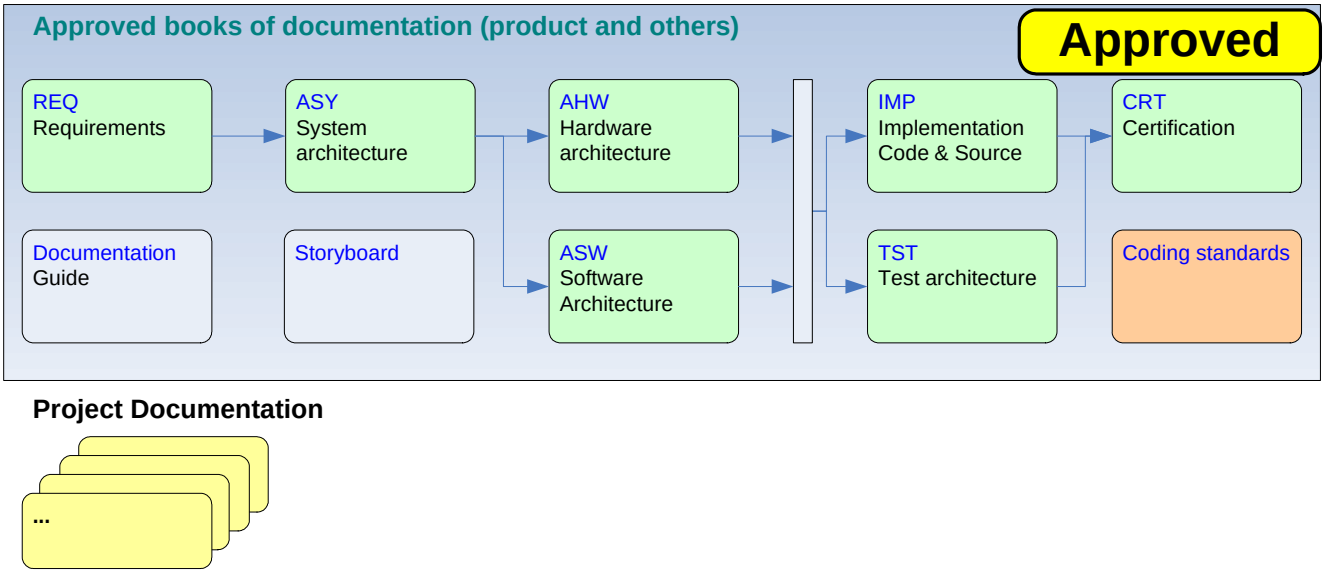


Figure 6-1. Documents with confirmed quality according to ezRead

6.1 Style recommendations

Good documentation style somewhat depends on the *availability of tools* and significantly depends on the *skill of author's* that understand the view of the reader. Some helpful hints are given below.

> The methods suggested below do have a distinc focus on the use of Adobe FrameMaker being a much superior tool than MS-WORD. However the general ideas may be taken to other editors, including MS-WORD.

6.1.1 Using standardized paragraph styles

ezRead suggests a set of powerful paragraph styles. Use those styles and do not invent your own styles. Using standard styles will essentially help for changing the *layout of books*, when required later.

If you cannot avoid adding a paragraph style, then consider to add it in The FormatRef.FM file and import into your document. This will at least allow to use the new style in other chapters before a similar style is invented there under another name.

6.1.2 Using highlighting techniques

Writing text, use *highlight* methods to help the reader finding an anchor of your statement. Ever got lost in long sentences? If so, *highlighting* would at least help to better catch the idea of the text flow.

Highlighting should be *consistent*. Do not exaggerate using **too many** different *highlighting* **techniques** (of course, this is an extreme example) - readability would drastically suffer. Make economical use of **bold** statements, they *will* catch the reader's attention, but if not really relevant, *italic* style is less disruptive.

Throughout ezRead documentation, *hyperlinks* and *only* hyperlinks are displayed in the color blue. Unlike often used, the hyperlinks are not underlined, because with many of them in good documentation, the readability would suffer too much.

6.1.3 Using hyperlinks (Cross-references)

Using hyperlinks (cross-references) is one of the basic features of ezRead documentation.

Note: The difference between hyperlinks and cross-references is of technical nature, semantically they are the same. (→ 3.3.2 "Cross-Referencing aspects" on page 3-26). In the following we will just use the notation "hyperlink".

Typically there are *three indicators* to use hyperlinks.

1. Link to an *overview* of the current topic to get the big picture
2. Link to *more detail* of a topic
3. Link to *information elsewhere* explained and not to be repeated in your text

You should find a good balance for repeating information compared to using hyperlinks. If text is essentially relevant for the reading, it is not a good idea to hyperlink because this could distract the reader from seeing things *in one picture.* If *complimentary information* or a *choice* [more detail / the big picture] can be offered, it is an excellent idea to use hyperlinks.

Hyperlinks to *external* documentation (PDF) shall be done using Preparing books for the final release and the methods described in that chapter.

> Using named destinations in external PDF documentation allows you to jump *right into chapters* of the target. You should make extensive use of this powerful technique. Readers will be most grateful for saving a lot of time through instant localization of the target. To apply this chapter-oriented link, it does not require more than described in 6.6 “Preparing books for the final release” on page 6-80.

Another really good idea is using *hyperlinks on abbreviations*. By far too many abbreviations are used in technical documentation that a “non-insider” cannot understand. And it is even worse. Very often abbreviations are ambiguous.

> *ECC* can mean “elliptic curve cryptography” or “European Citizen Card” and if “*ECC is used in ECCs*” it gets complicated to make out the difference. Although you cannot solve such collisions, you can use cross references to refer to Abbreviations or to items in a Glossary.

Even common expressions like URL can be helpful to be explained at little additional cost. To better find the expressions in the list of referable paragraphs, *two special paragraph styles* have been provided for the Glossary: *TagAbbrev* and *TagGlossary* which collect all items of the glossary/abbreviation list.

6.2 Using Graphics

“A picture tells more than a thousand words”

Good documentation makes use of figures/graphics Since the in-built graphic capabilities in FrameMaker or other editors are small in contrast to specialized graphic tools (e.g. VISIO) graphics have to be imported. An exception to classical export/import is the *object import* as shown in the next chapter.This is the preferred choice, when available.

6.2.1 Include time stamps to architectural images

Every architecture diagram shall contain a version information or *last modification date*, usually by stating the modification date.

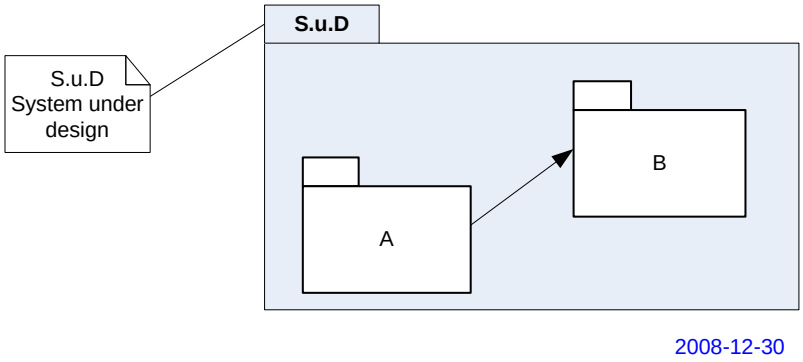


Figure 6-2. Creation datestamp on figures

Screen shots do not require timestamps since they typically do not qualify as architectural units.

6.2.2 Using VISIO graphics

Technically, VISIO graphics can be included according to “Import as an Object” in volume 2 page 8-1).

Always store the source appropriate graphic in a directory close to your document. The recommended sub-folder is “VSD”.

However, is has been found, that *when using particular VISIO objects* (e.g. a Server image), Framemaker has problems to *print* the final document or books. Therefore it is highly recommended first to export a VISIO drawing as WMF file and then import the appropriate WMF file as graphic in FrameMaker. This has two advantages

- FrameMaker documents (.FM) open much faster because they do not need to invoke MicroSoft VISIO when showing graphics
- A VISIO drawing can maintain an automatic field with the “current date”. Whenever the VISIO is changed an exported to a WMF file, then this date will automatically be fixed in the WMF file to represent the date of the last modification.

Warning: Never include drawings of any nature as OLE link *only.* (typically at the comfort of *cut and paste*).Such bad use would make the complete transport of source files hard or impossible.

6.2.3 Using Image import

Other graphics can of course be imported. Technically this is described in “Import as a Graphics File” in volume 2 page 8-4.

6.2.3.1 Importing after export from other programs

For the import/export, different formats are available. Whenever possible, use *vector based* export formats like WMF, EPS, EMF. *Vector based formats* are formats that do not store the pixels of a drawing (like in a photo) but drawing primitives (lines, circles, etc.). In particular - *written text* in vector graphics will perfectly scale - opposite to poor appearance of fonts when exported as pixels.

For Enterprise Architect Use WMF export format to achieve vector quality.

6.2.3.2 Using Screen Shots

Screen shots are always pixel based formats and they are best stored as PNG files. The PNG format is a modern compression format for bitmaps that works without information loss allowing transparency features. When importing PNG files from screen shots according to “Import as a Graphics File” in volume 2 page 8-4 use **96 dpi** to get proper scaling.

6.3 Using formatting templates

Formatting templates help to maintain a corporate design throughout different books. This is a particular Adobe FrameMaker idea, but similar methods could be applied when applyting style sheets to MS-WORD documents e.g. by the "Normal.dot" for style import.

6.3.1 The FormatRef.FM file

As said in 6.1.1*standard paragraph styles* should be used. All styles are available in the *...\XX00_Templates\FormatRef.FM* reference file. All documents in the book(s) of a project will import their formats from *FormatRef.FM*. This allows to easily apply different layouts to the books by changing *FormatRef.FM* only and let the other files copy the paragraph styles from it.

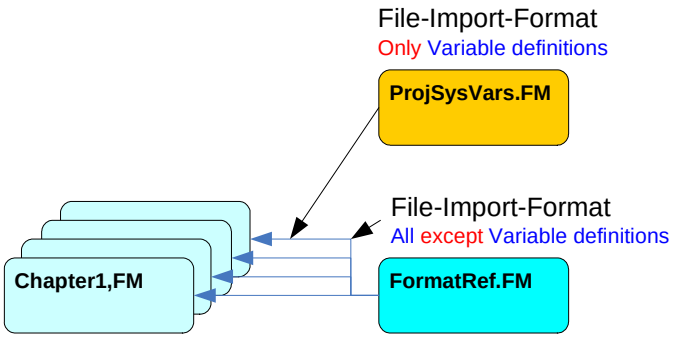


Figure 6-3. Import Formatting and Variables

In order to import formats, first Open the **..\XX00_References\FormatRef.FM**.

Then select the BOOK tab 

and select all files of the book

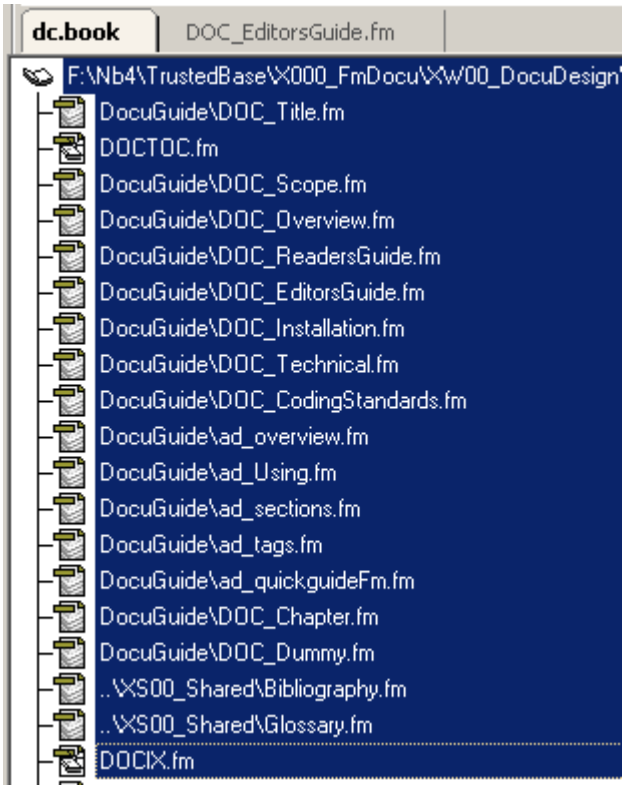


Figure 6-4. Select all files in the book

Select the FrameMaker "*File-Import-Formats*" menu and select the formatting import file **..\XX00_Templates\FormatRef.fm** as source ("*Import from Document*". Then check all fields but the "*Variable Definitions*", Variables will be imported from The SysVars.FM file which is project specific in contrast to the The FormatRef.FM file.

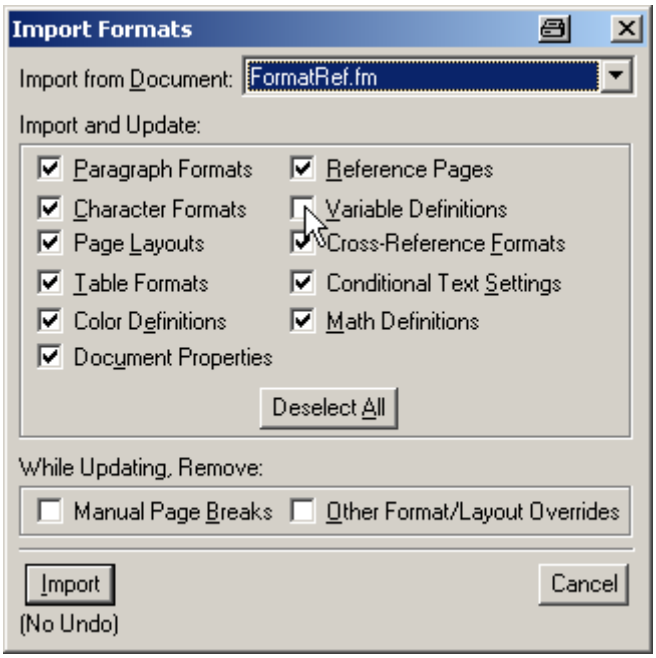


*Figure 6-5. Select all **but** Variable Definitions*

After *Import*, the paragraph styles of *all selected documents* in the books will be updated.

6.3.2 The SysVars.FM file

System variables are typically *project*- or *book*-specific, whereas document/paragraph formats are often *company specific*. Therefore the system variables are held in a separate project specific document **..\XX00_Templates\SysVars<proj>.fm** where **<proj>** is any project specific abbreviation (e.g. **DOC**, **TBOS**, **CAMS**).

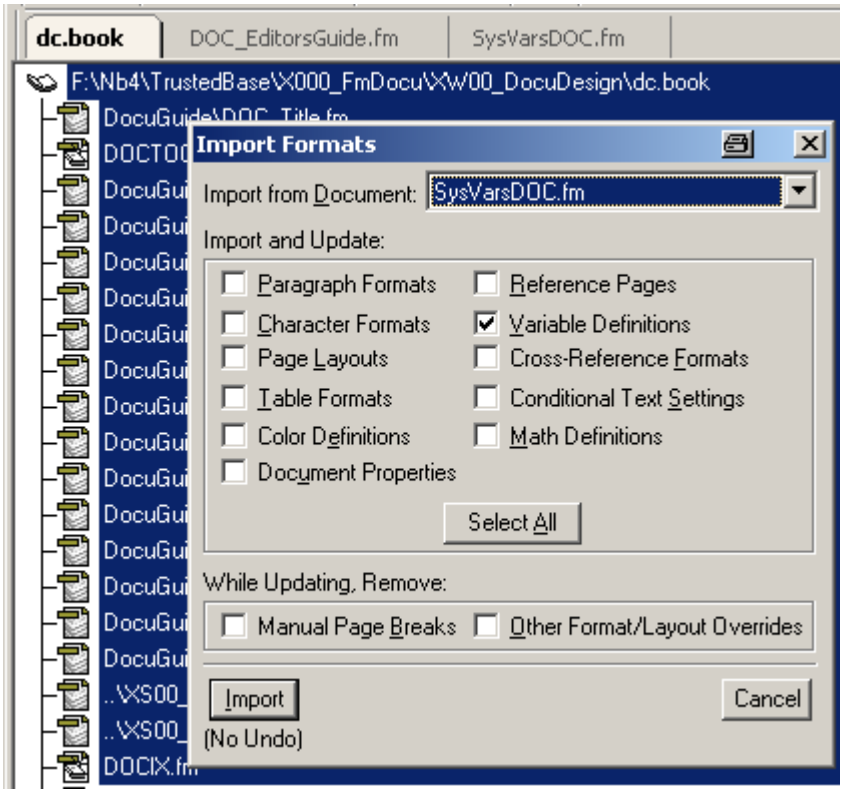


Figure 6-6. Import Variable Definitions

Proceed according to "The FormatRef.FM file" but select the *Variable Definitions* as only parameter to be imported.

6.3.3 Don't care for document layouts

It is not important to create special character style options like overlining chapters unless you have a very local requirement. Typically style elements should be applied generally as discussed in The FormatRef.FM file.

6.4 Where to put documents

ezRead proposes the following directory structure

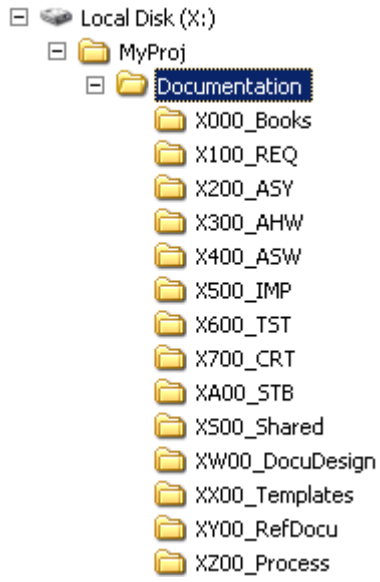


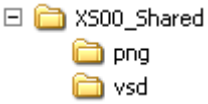
Figure 6-7. ezRead Directory structure

X000_Books: stores all *final release*. BOOK files involved in the documentation tree. Find more about the idea behind X000_Books in 6.6 “Preparing books for the final release” on page 6-80.

X100_REQ .. X700_CRT: stores the source files (.FM) and the development books (.BOOK) according to 4.1 “Book organization” on page 4-35.

XA00_STB: stores the Storyboard as described in 4.3.1 “Storyboard” on page 4-44.

XS00_Shared: stores those files shared between all books. Those files are typically the *Glossary*, the *Abbreviations* and the *Bibliography*. The directory should also keep common *graphic resources* like VISIO, WMF, PNG or JPG files.



It is recommended to sort graphic files by their file type as shown above.

XW00_DocuDesign: stores the Documentation Guide or at least those files of a standard documentation guide, that are project specific. The files correspond to the idea of 4.3.2 “Documentation Guide” on page 4-44.

XX00_Templates: store the reference files as explained in “The FormatRef.FM file” and “The SysVars.FM file”.

XY00_RefDocu: is the only directory which is subject to be exported. Any external PDF file of the project shall be stored in XY00_RefDocu.

It might occur to the reader that keeping files in XY00_RefDocu is an invitation to keeping documents in two places which is often considered bad practise. Indeed double-sourcing is the intentional case here, but keeping a copy of relevant external files is wanted

- to isolate the documentation tree from the tides of document changes “out there”
- to maintain additional processing features on those documents like the creation of bookmarks and Preparing books for the final release.

The *encapsulation of reference documentation* is a vital factor to avoid collisions with other development activities. The additional synchronization work is wanted for the sake of controlled *content oriented* version maintenance (in contrast to file-oriented).

XZ00_Process: often holds the documentation project working directory. Unless otherwise suggested in the project, XZ00_Process is recommended to be used as documentation environment (→ 11.7 “Set/Clear documentation environment” on page 11-178).

6.5 Using meta-tags

The idea of Meta Tags Strategy is explained in 3.3.5 “Meta Tags Strategy” on page 3-28.

The same meta tags can be used in

- MS-WORD documentation
- Source Code

Developers can accept cross-referencing technique much better if it is easy to handle with meta-tags.

Meta-tags also cover *version handling aspects.* A reference document might be updated or its file name be changed. Unless there are significant changes in the structure, the meta tags can simply be re-processed to comply with the new version, whereas *direct references* (like done in doxygen) would require more complex update strategies. More on this can be found in 6.7 “Version management” on page 6-81.

The most frequently tags used in MS-WORD and *source code comments* are

- <xref
- <v
- <url
- <ND
- <ix

In *source code comments* the following tags are recommended in addition to those above

- <fig
- <tab
- <xmp>, <exmp>
- <eq

6.5.1 <xref

Syntax

<xref *FileID_NamedDest*>

Description

FileID_NamedDest is a concatenation of a *file identifier* and a *named destination* (→ Preparing books for the final release).

The *FileID* associates a file with its entire path and file name. Developers will learn and remember those relevant *FileID*s if their names are reasonably chosen by Editorial Team. A list of abbreviations shall be stored under the XX00_-Templates directory.

> The Editorial Team maintains a list of abbreviations which match a particular file. In the Example below, the identifier DBSPEC could associate the file “*...\XY00_RefDocu\DatabaseSpecification_4v51.PDF*”. When processing meta-tags, the Editorial Team’s tools resolve those references. This tag also complies with the BKX tag according to 12.1.10.1 “BKX:Bookmark CSV File structure” on page 12-194).

The *NamedDest* should match an actual named destination in the target document. Such it is formed according to the rules specified in 7.1.4.1 “Step 1: Choose the right destination name”.

Chapter destinations are available in all documents maintained by ezRead. If an author desires to address a particular point, he should ad the named destination in the target document according to 7.1.4 “How to create a named destination manually”.

Example

Find more detail in **<xref** DBSPEC_Ch4p7>.

6.5.2 <v

Syntax

<v *VariableName*>

Description

VariableName is the name of a variable. The idea for this tag comes from auto-documentation in source code where it is often important to highlight a variable name for better readability in the final documentation.

But also when processing MS-WORD documentation this tag can be used for later processing. In MS_WORD, alternatively a *BPv* character style can be established to indicate variables.

The advantage is, that the actual highlighting technique can be chosen later (e.g. colored or style).

Example

The variable <v *dbEntriesRemain* >. stores the number of available database entries.

6.5.3 <url

Syntax

<url *www.google.de*>

Description

<url > introduces a URL, typically a Web address or a mail receiver when used like <url *mailto:helmut@hscherzer.de*>

Example

For more info refer to **<url** *www.wikipedia.com*>.

6.5.4 <fig

Syntax

<fig *FigureReference*>

Description

The **<fig** tag is used to insert figures into processed documentation. *Figure Reference* is a unique identifier for a figure in a particular Framemaker library file. The tag can be used in MS-WORD documentation that is created as input to one of the books, but is typically used in *auto-documentation* situations of source code.

In source code, it is impossible to include figures in an ASCII file. Authors of source code may user the **<fig** Tag in order to address a link to a graphic file.

The actual figure will be placed in an *ordinary FrameMaker file* using a *caption text* with **<Figure** FigureReference>. Auto-documentation tools will visit the references file and find the *FigureReference* in the caption. Processing the source, the tools will insert the figure right where the programmer placed the link.

Example

A program source code could look like

```
PVOID MemAlloc(int size)
{
  // ----------------------------
  // @step 1 | PrepareAlloc |  The following figure demonstrates the
  //           allocation sceme <fig fgMemAlloc>
  Handle = MemPrep(size);
  ...
}
```

The FrameMaker reference file would have included at any place

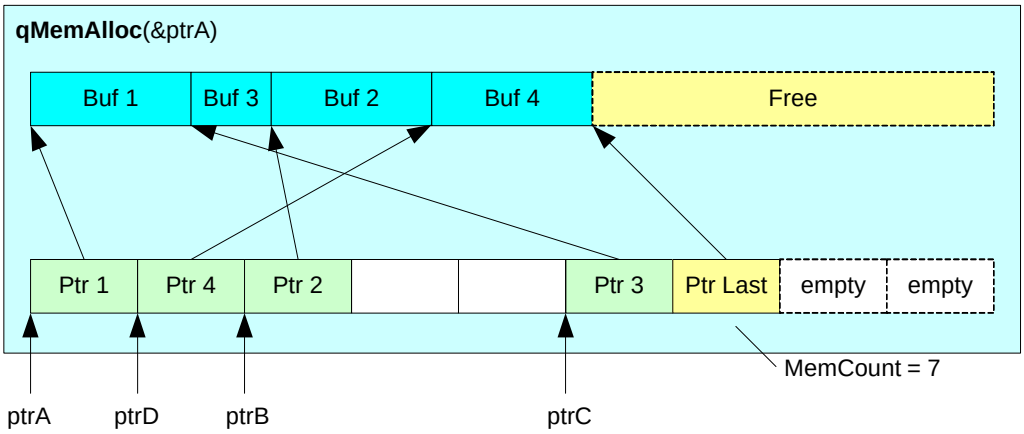


*Figure 6-8. <Figure fgMemAlloc>Double de-referencing of **qMemAlloc** pointers*

The final auto-documentation text would show up as

4.7.3 Step 1: Prepare Alloc

The following figure demonstrates the allocation scheme

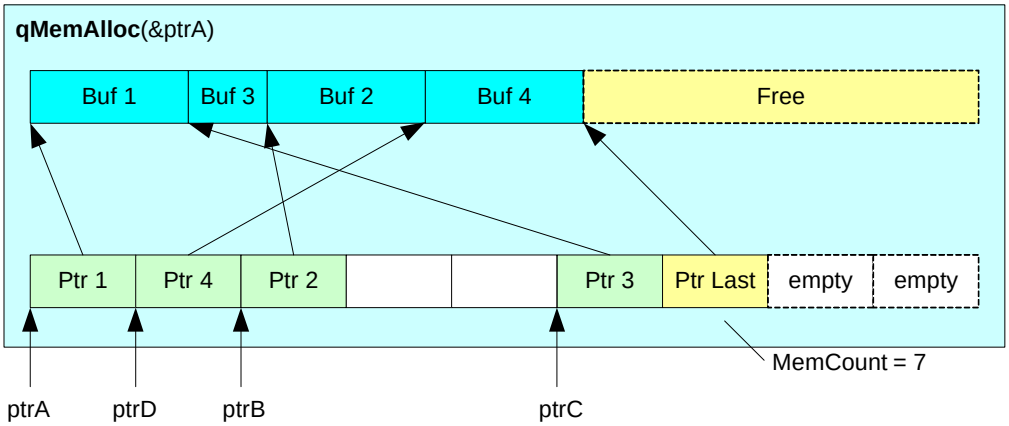


*Figure 4-1: Double de-referencing of **qMemAlloc** pointers*

6.5.5 <tab

Syntax

<tab *TableId*>

Description

Along the lines of the <fig tag the **<tab** serves a similar purpose. In ASCII source files it is difficult to write tables. Using exactly the same methods as with the <fig tag tables can be included. Instead of a figure with caption <Figure, a *table* should be created with the caption **<Table** TableId>.

Example

A program source code could look like

```
int Select(int Pos)
{
  // ----------------------------
  // @step 1 | SelectEntry |  The following table shows all possible
  //           entries available. <tab tbSelectTable>
  Handle = TableSeq(Pos);
  ...
}
```

6.5.6 <eq

Syntax

<eq *EquationID*>

Description

Along the lines of the <fig tag the <tab serves a similar purpose. In ASCII source files it is difficult to put equations. If, however required for documentation, by using similar methods as with the <fig tag, equations can be included. Instead of a figure with caption *<Figure*, an *equation* should be created with the caption **<EQ** EquationId>.

Example

A program source code could look like

```
float SqRoots(float a)
{
  // ----------------------------
  // @step 1 | Solve Roots | The roots are solved by the following
  //           algorithm scheme. <eq SolveRoots>
  Temp = p / 4;
  ...
}
```

The FrameMaker reference file would contain at any place the *Equation* style with numbering parameter SolveRoots.

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q} \qquad \text{SolveRoots}$$

After *auto-documentation* processing, the result would read

> **4.7.3 Step 1: Solve Roots**
>
> The roots are solved by the following algorithm scheme.
>
> $$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

6.5.7 <ix

Syntax

<ix *TopicName*>... simple construction
<ix *MainTopic : TopicName*>... with main topic
<ix *TopicName1 ; TopicName2*>... two different aliases
<ix *MainTopic : TopicName1 ; TopicName*>... combine main topic and aliases
<ix *TopTopic : MainTopic : TopicName*>... even three levels of nesting

Description

The *<ix* tag generates an index marker.an a separate index chapter. Using the *<ix* meta tag allows automatic generation of an appropriate index by the Editorial Team.

Example

Using an index entry *<ix Code>* will generate an entry in the index that shows up like

```
code ........ 70
```

or *<ix Explanations:code>* would show like

```
Explanations
   code ..... 70
```

Further information on formatting options can be found in the FrameMaker User Guide Chapter 12.10 on page 435.

6.5.8 <ND

Syntax

<ND *NamedDestination*>

Description

The **<ND** tag positions a named destination at its occurrence. Read more about Preparing books for the final release in 6.6 “Preparing books for the final release” on page 6-80.

Example

Within the comment of a program or a MS-WORD file you can use

<ND Test32p5>Test 32.5 - Tests the temperature behavior of the system

6.5.9 <xmp>, <exmp>

Syntax

<xmp>
... text ....
<exmp>

Description

<xmp> and <exmp> mark a region for example text. Mainly used in source code commenting these tags indicate when further processing should use fixed pitch fonts and suspend auto-break adjustment in order to show an example exactly as written.

Example

```
The following pseudo source code shows the
data flow for the mailing process
<xmp>
  if Exists(MsgBox)
  {
    for (MsgCount = 1; MsgCount < NoMessages; MsgCount++)
    {
      ProcessMail(MsgInBox[MsgCount]);
    }
  }
<exmp>
where <v MsgCount> is the 0-based index into <v MsgBox>.
```

will produce the text

The following pseudo source code shows the data flow for the mailing process

```
if Exists(MsgBox)
{
  for (MsgCount = 1; MsgCount < NoMessages; MsgCount++)
  {
    ProcessMail(MsgInBox[MsgCount]);
  }
}
```

where *MsgCount* is the 0-based index into *MsgBox*.

6.6 Preparing books for the final release

The X000_Books directory stores all BOOK *final release* BOOK files involved in the documentation tree.

All BOOKS must be open

To generate the final PDF of a BOOK, *all other books* that are referenced from any of the book's chapters, shall be opened too. This has a technical reason. When *resolving cross references* of a book, FrameMaker requires to have all referenced books *being open* in *that* directory where their corresponding PDF file will be found after all books are processed.

If all BOOK files reside in X000_Books, then all PDF files are expected to be found in the same directory. And that is good for export. For exporting files, however, a similar technique is available (→ 7.2 "Exporting books and references" on page 7-110).

How FrameMaker resolves PDF links

If BOOK files would reside in *different* directories, the corresponding PDF files would have to be created and kept in those directories too. Only then the cross references would resolve correctly. This is bad for export because those directories would have to be delivered and maintained by the receiver.

For the delivery of documentation, it is wise to keep the complexity of the directory tree at a minimum. Therefore the *final release books shall be organized in one directory* (X000_Books) *only*. For the same reason, *all other references* (external PDF files) will have to live in XY00_RefDocu.

> The TOC and IDX files must technically be in the same directory as the BOOK. This is a hard FrameMaker requirement.

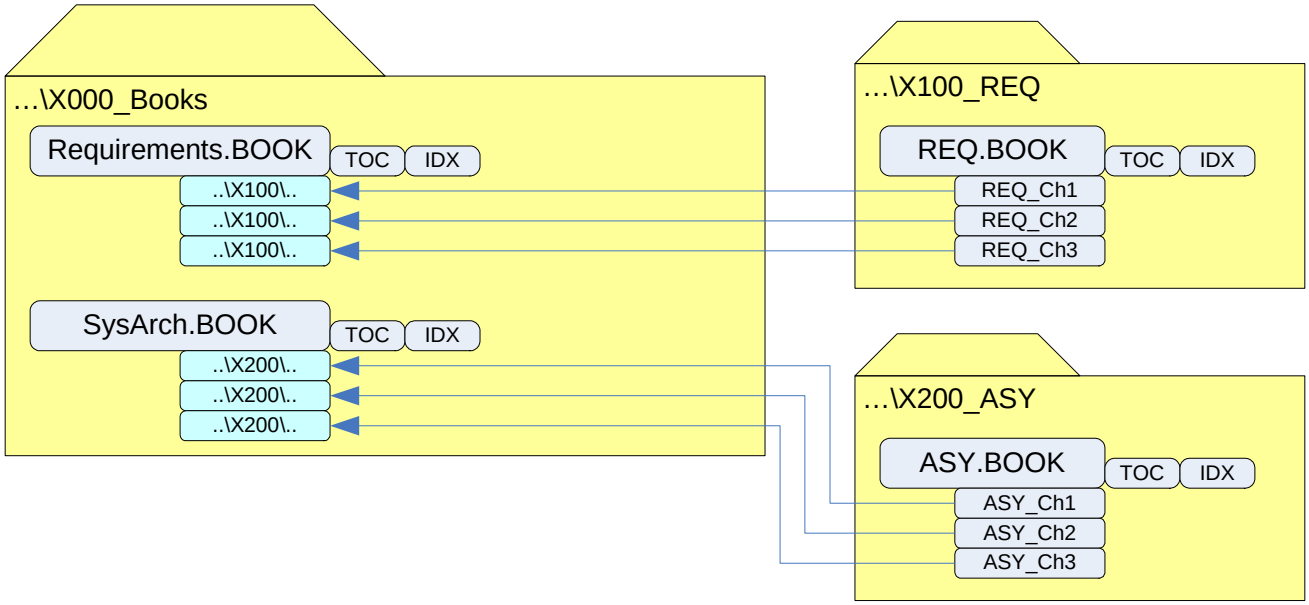


Figure 6-9. Store release books in X000_Books folder, keep the source files in separated folders

X000_Books is to be seen as a *release* directory. While the books are under development, it is even recommended to keep *development book files* in their *local* directories. This is reasonable to the aspect of *document ownership*..

> A book writer might want to add some special files to the development book with information that (s)he would not want to have in the final release. Two typical examples are The FormatRef.FM file and the The SysVars.FM file, and often a *Dummy.FM* is held also as *Save-As-template* for new chapters. Having those files in a *development* BOOK file allows faster access when those files are needed (e.g. to create a new system variable in SysVars__.FM).
>
> For the release, those helpful additions in the book are not wanted, they would spoil the generation of a table of contents by unwanted chapters.

Conditional processing

Another reason to maintain release books separately, relates to *conditional processing*. A book that is delivered to a customer should perhaps exclude some chapters with *internal* information about *undocumented features*. Within chapters, conditional text processing is possible to exclude unwanted details to the customer.

Views on books

On release, conditional settings can be set based on BOOK files. This allows to maintain several views of books and allows the delivery to most diverse receivers. The different book files for those diversification shall live in X000_Books whereas a document developer wants to see *everything* and can put the conditional settings in his/her local book accordingly (*Special - Conditional Text - Show/Hide ...*)

6.7 Version management

The most popular question when presenting the idea of Cross Reference Techniques is: “But what happens if referenced documents change - will I have to redo my cross references?”.

Unless this question is used politically as an argument not to put in cross references at all, luckily the ‘damage’ from moving targets can be held low. Moving targets are always associated with some rework but in the following we give some best practices to minimize this additional work.

First of all we distinguish between targets of (rather) static nature and those that are really subject to frequent changes.

6.7.1 Static targets

Typical representatives for static targets are *international standards*, *vendors specifications*, *Data sheets* or *fixed versions* of reference material. Very often they are received as PDF from vendors or institutes. These documents do have a rather long lifetime (years) and are not exposed to many changes after their publication. For instance a paper once published on a conference is unlikely to *ever* change after.

6.7.2 Dynamic targets

Typical representatives for dynamic targets are documents presently under work or subject to frequent rework. Several teams working out an architecture of a product will see many changes until a given deadline. Generally the more access authors do have on a document, the higher the risk of changes will be.

6.7.3 Target tagging techniques

We distinguish between

- Chapter number oriented referencing

- Named Tag oriented referencing.

While in the final PDF both of them will technically end up in Hyperlinks to named destinations the Best practices to overcome the 'moving target'-Problem suggest a different approach depending on references to static or dynamic targets.

6.7.4 Chapter number oriented referencing

References are made to chapter numbers. Technically the references should be to named destinations that were likely to be created from bookmarks or other document artefacts (→ 7.1.5 "How to create named destinations automatically" on page 7-92).

Referring to chapter numbers is elegant, but has the disadvantage, that on the insertion of a Head *N* chapter, references may be wrong until the next occurrence of Head (N-1) in the target document.

There is no elegant work-around, although it would technically be feasible to change references in client documents, it is not wise to let tools touch an entire tree of files on this purpose.

The big advantage of chapter number oriented management is, that the chapter targets (named destinations) are very easy to extract in target documents.

6.7.5 Named Tag oriented referencing

Named Tags are anchor points in target documents, that have a name. The name is recommended to be given according to the rules specified in 7.1.4 "How to create a named destination manually" on page 7-88.

In MS-WORD, these named tags can be created through bookmarks (→ 8.1.4 "Link to external MS-WORD document with DOC bookmark" on page 8-122.

In FrameMaker the creation of an explicit named destinations is also possible (Special - Hyperlinks - insert hyperlink), but not necessary as FrameMaker offers more sophisticated techniques to solve the versioning problem (→ next chapter).

The advantage of named tag referencing is that the named tags move along with the insertion and relocation of chapters. The disadvantage is, that named anchors are by far not as intuitive and easy to find as chapter numbers are.

A special form of generating bookmarks and named destinations from PDF comments is described in 12.1.8 "Generate named destinations/bookmarks from comments" on page 12-191. Those can be helpful in particular if the target PDF file is more likely to change (→ 12.1.8.3 "Version changes in target documents" on page 12-193)

6.7.6 FrameMaker referencing

Using Adobe FrameMaker, however, solves the problem in the most elegant way. Whenever Framemaker visits a target chapter (section/paragraph) first, it (internally) computes a unique hash value on the target and registers the hash value on the referring document. If the hash value, however, is already present, the existing hash value will be referred from all other files that want to point to the chapter.

This technique combines the fast visibility of Chapter number oriented referencing and the flexibility of Chapter number oriented referencing but again - only works on targets that are under the governance of the same author and that are written in FrameMaker.

6.7.7 Best practices to overcome the 'moving target'-Problem

The best practices help to find the right strategy for working with Cross Reference Techniques.

Using static targets: For Static targets use Chapter number oriented referencing. Named destination can be easily generated from bookmarks (→ 7.1.5 “How to create named destinations automatically” on page 7-92) and bookmarks typically address chapters. As static documents are unlikely to change, the easiness of generating the targets makes this choice favorite. Also chapters are better to find than named tags (destinations).

Using dynamic targets: For Dynamic targets use Named Tag oriented referencing. The named tags will move with the insertion and relocation of chapters and sections. This comfort, however, has to be paid with the fact, that named tags (e.g. MS-WORD bookmarks) are not so intuitively recognizable as chapters are. And of course, they cannot be generated automatically from bookmarks.

Using FrameMaker: Just by using cross-references you don’t need to care for the moving target problem since FrameMaker solves this implicitly by the way it creates cross references. (→ [FmUsr#5] and Chapter 6. “Cross-References” in volume 2 page 6-1). FrameMaker uses *on-first-creation* random tag values in order to internally tag a cross reference which might change its chapter number and title.

Other editors (e.g. MS-WORD): Other editors also support links to moving targets, they typically provide an internal bookmark. which stays constant even when changes occur.

6.7.8 Content oriented version management

Content oriented version management is in contrast to *file-oriented* version management. It carries the idea, that it can be recognized when new external references are to be included or if implementation changes and change requests need to be reflected in the product documentation.

The recognition of content changes can be done automatically by specials tools written for this purpose, or manually by simply detecting changes and investigation by author(s). Again tools can assists authors on this work.

What to do on Version changes

In order to integrate changes into the product documentation it is not enough to replace a versioned document, but to check for *content differences* and think about the impact of those to the entire set of books. Many times, the effort is little, sometimes it is hard. If it is hard to update the documentation, then it is even more important to reflect system changes in the documentation.

The *detection* of changes is done easily since it is not allowed to anybody but the Editorial Team to add a document to the reference tree XY00_RefDocu.

7 Cross Reference Techniques

7.1 Using Cross References and hyperlinks 85
7.1.1 Cross References 85
7.1.2 Hyperlinks to named destinations 85
7.1.3 How to view named destinations in a PDF file 86
7.1.4 How to create a named destination manually 88
7.1.5 How to create named destinations automatically 92
7.1.6 Stub technique 101
7.1.7 Content based destination generation 102
7.1.8 How to enumerate bookmarks 105
7.1.9 Enumerate – Merge function 107
7.1.10 How to jump to a named destination 107
7.1.11 Linking FrameMaker to SECURED documents 108
7.1.12 Instant Jump to Stub 109
7.1.13 Instant Jump with dialog box 109

7.2 Exporting books and references 110
7.2.1 Using the ND.API Export function 110
7.2.2 Target directories 112
7.2.3 References from Books to Books 114
7.2.4 Missing references 115
7.2.5 Clearing the export directory 116

7.3 Regular expressions 116

7.1 Using Cross References and hyperlinks

ezRead makes extensive use of hyperlinks. (see 3.3.2 “Cross-Referencing aspects”).

- Cross References
- Hyperlinks to named destinations

7.1.1 Cross References

Cross References – in contrast to Hyperlinks – are links from a FrameMaker document to another Framemaker document. The creation of internal cross references is well described in Chapter 6. “Cross-References” in volume 2 page 6-1 and will not be considered here furthermore.

It shall be reminded, however, that on final PDF production of a FrameMaker document, all BOOKS being referenced, have to be opened before PDF production in order to let FrameMaker resolve the references to the books instead of the chapter file.

7.1.2 Hyperlinks to named destinations

Hyperlinks are much like cross-references, however they do not link to FrameMaker documents but to other kinds of documentation and also URLs. In PDF files *Named destinations* are targets of hyperlinks, in MS-WORD these targets are called "bookmarks".

Why named destinations?

Named Destinations are the only official way to open a PDF file by a named location in contrast to a "page". In particular a named destination will open a PDF on a particular location of a page (where the named destination was set) – opening on a page will always open at the top of a page.

Named destinations can be added at any place in an existing PDF file. Giving them a unique *name* allows a very efficient setup of hyperlinks.

The advantage is, that those named destinations can later be added to *external documentation*.

Manually adding named destinations to many external (PDF-)documents would be very time consuming and not affordable. Therefore a *set of functions* was created to *auto-generate* the most important destinations - typically the chapters of a file. This very often reduces the time for the creation of named destinations to a few minutes or less.

The functions are explained in 12.1 “The ND.API plug-in” on page 12-183 and in the chapters below.

7.1.3 How to view named destinations in a PDF file

In Adobe Standard/Professional you can *view* named destinations in a separate window using “View - Navigation Panels - Destinations”

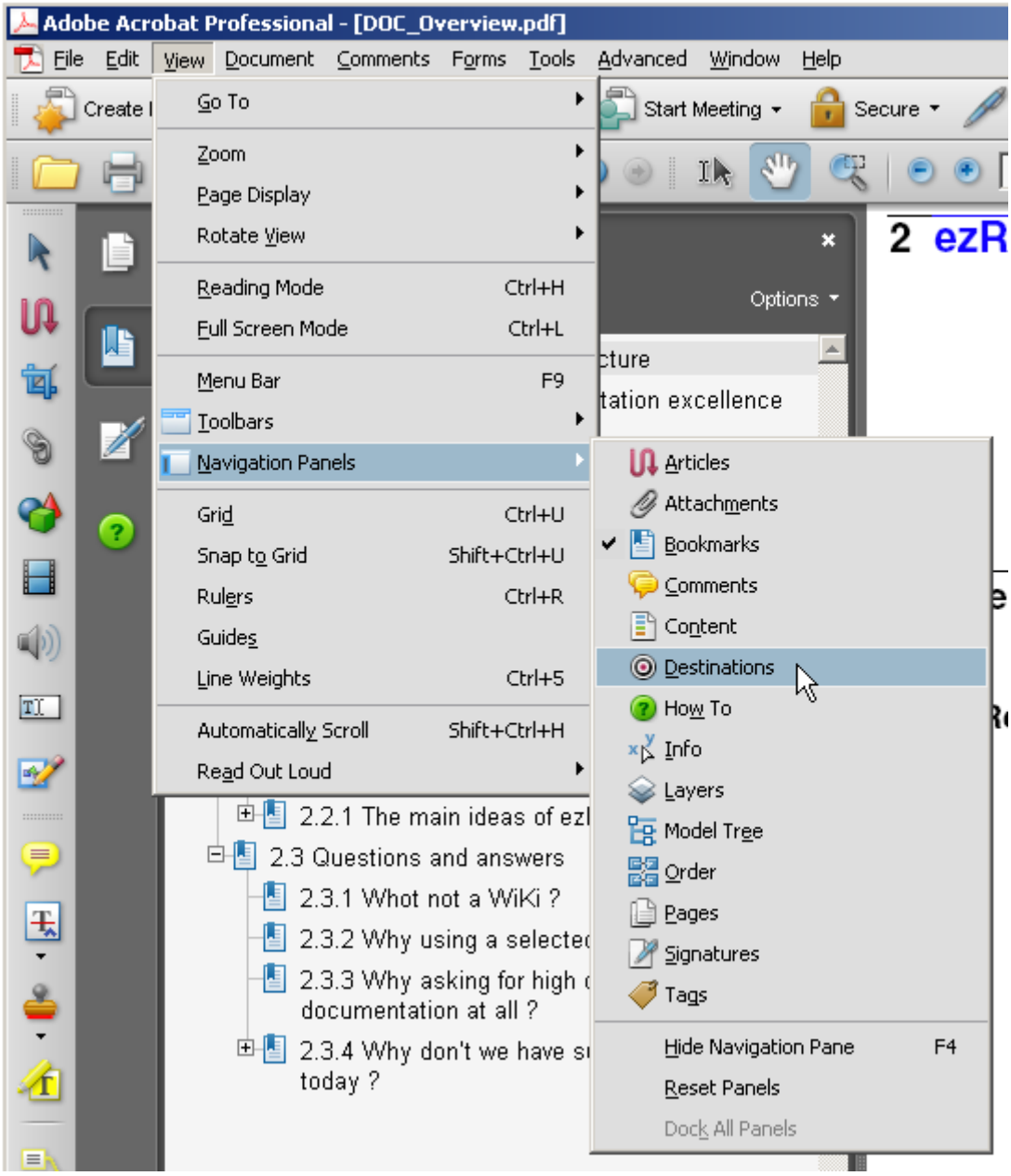


Figure 7-1. View named destinations in Adobe Acrobat Standard/Professional

A box will appear showing named destinations.

In Acrobat X and later, the function is found under

View – Show/Hide – Navigation Panes – Destinations

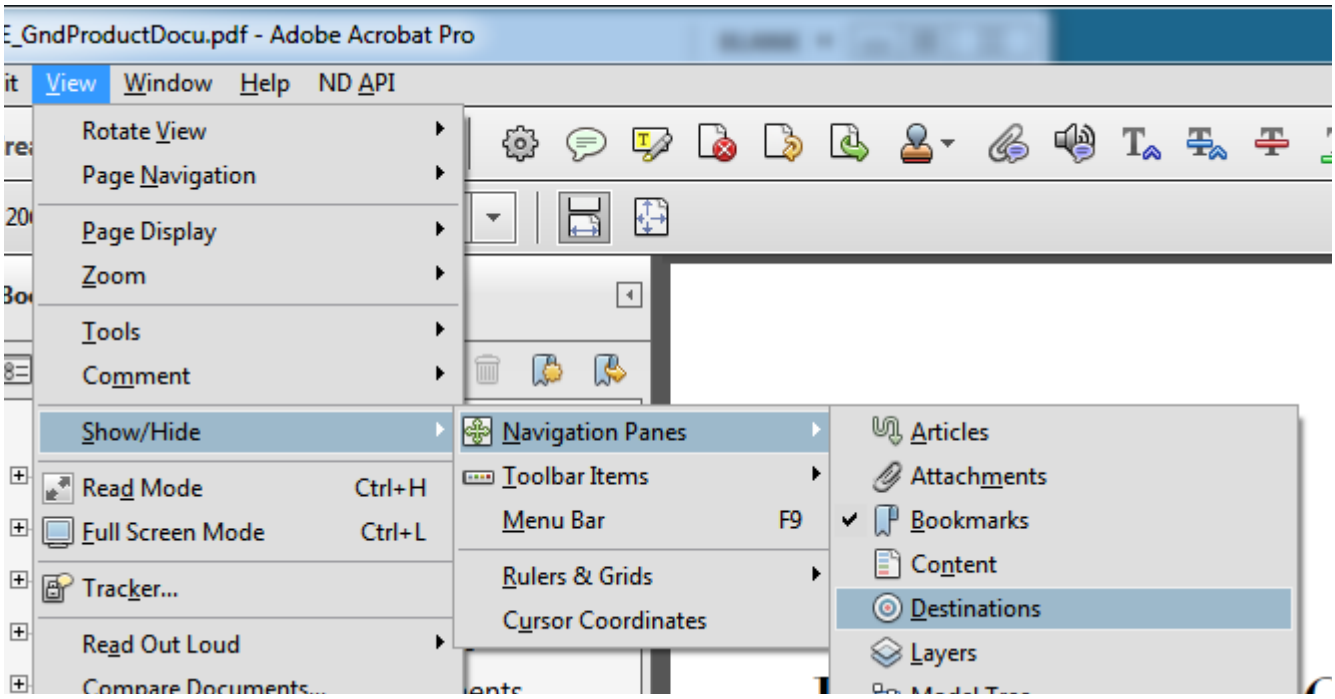


Figure 7-2. View named destinations in Adobe Acrobat X

The destinations will be shown in the associated view. The "Destinations" tab can be moved to the general bookmarks section to be permanently available. Move the tab only – not the entire window (which wouldn't work).

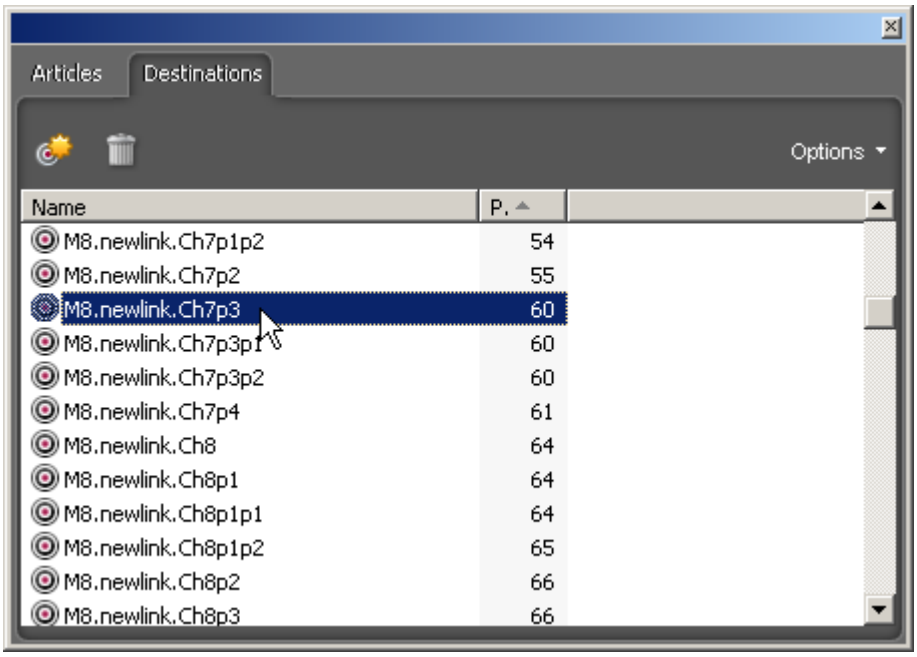


Figure 7-3. List of named destinations

The above box already shows some very special destinations. Typically all sort of entries like “F939082” or “G3.1032272” may occur. Those are named destinations that FrameMaker or MS-Word will generate automatically.

> Do not delete such odd destinations when you find them in a document. Some editor programs use named destinations for internal cross references. If you delete those destinations, the internal cross references (e.g. Table of Content links) won’t work anymore

The special thing about the destinations in Figure 7-3 is the prefix “*M8.newlink*”. This is a prefix generated and required by Adobe FrameMaker. Whenever you insert a hyperlink to a named destination in an Adobe FrameMaker document (see 7.1.4below) then FrameMaker will add the prefix “M8.newlink.” ahead of your desired name.

Example: Creating a link to a named destination “MyDestination” in a FrameMaker document will actually create a hyperlink to the named destination “M8.newlink.MyDestination”.

This convention is very practical. It allows to identify every “hand-made” named destination in a PDF file and appropriate tools will make a good use of that prefix.

7.1.4 How to create a named destination manually

Sometimes it may be necessary to create a named destination manually in a PDF target file. This can be the case, if you need to link at *higher granularity than chapters only*. If for instance you have a table and you want to guide the reader to a particular entry, then it is necessary to add a named destination manually.

A named destination is always added in the target document i.e. the PDF file into which you want to link. The named destination itself is the anchor point. The best way of adding a named destination manually is described in 12.1.2 “Set Named Destination”.

7.1.4.1 Step 1: Choose the right destination name

Whenever you choose a name for your named destination you have to understand some restrictions on named destinations.

Only *letters* and *numbers* are allowed to be used in named destinations. Trying to use spaces, dots, _ underscore or - dashes will make it impossible to link to a named destination of your choice. Adobe PDF technology uses these characters for other purposes and therefore these rules shall be followed.

Table 7-1. Characters in named destinations

Allowed	Forbidden
a..z, A..Z 0..9	_ (underscore) - (dash) . (dot) <space> and all the rest of non alphanumerics like ({[ ] }) etc.

Due to the limited character set we highly recommend the following techniques. Replace the forbidden characters according to the following scheme.

Table 7-2. Recommend notation

Forbidden char	Replacement	Example
. (dot)	by **p** (“point”)	Chapter 3.5.1 → Ch3**p**5**p**1
- (dash)	by **s** (“da**s**h”)	Close-Up → Close**s**Up
_ (underscore)	by **u** (“**u**nderscore”)	BX_Unit → BX**u**Unit
<space>	by BiCapitalization	system unit → **S**ystem**U**nit

The above conversion is quite easy to remember and comes at an affordable readability.

7.1.4.2 Step 2: Position your document

A named destination, when linked to, will always be placed the the y-position at the top of the window.

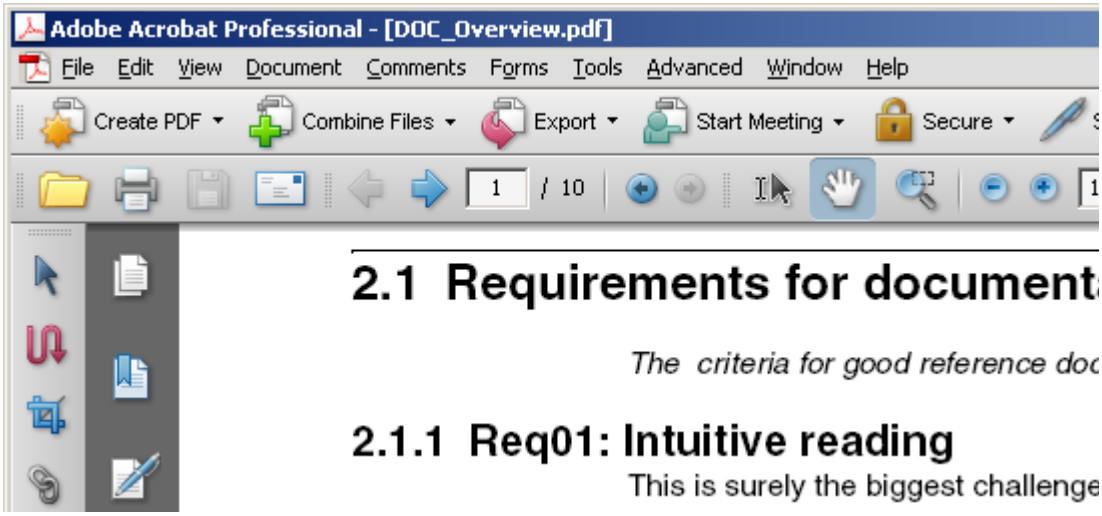


Figure 7-4. How a named destination positions

In the above example, Chapter 2.1 contains a named destination. When hyper-linked, Chapter 2.1 will be placed at the top of the window. The positioning is configurable as a function of The ND.API plug-in. See under 12.1.23 “Set ND view reference” on page 12-207.

7.1.4.3 Step 3: Choose the zoom factor

Named destinations do not only store the position but also the zoom factor at the time of their creation. It is recommended here to use the “*Fit Width*” setting (use *Ctrl-2* or *View - Zoom - Fit Width*).

7.1.4.4 Step 4: Set named destination using the ND.API plug-in

The ND.API plug-in allows a comfortable entry of a named destination which avoids the window in the next step.

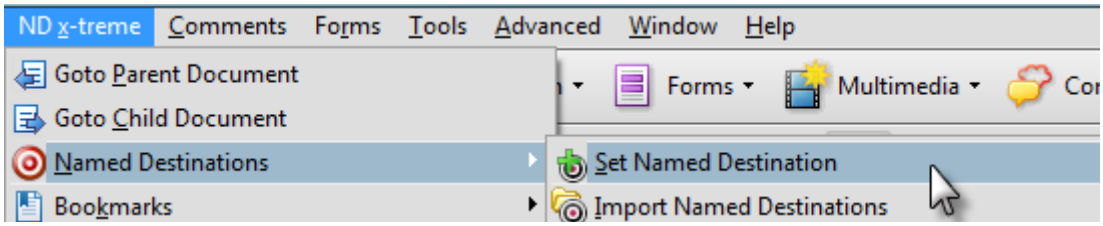


Figure 7-5. Entering a named destination

Use - Advanced - Manage Destinations - Set Named Destination open the following dialog box:

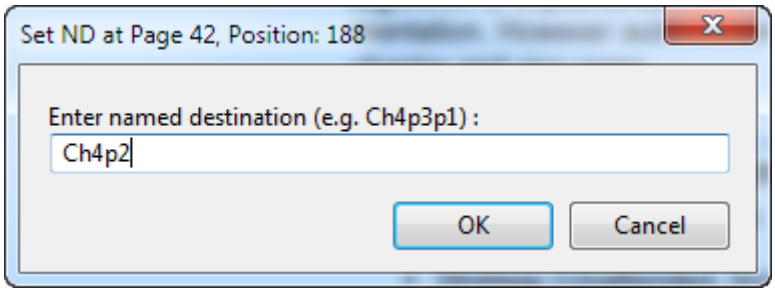


Figure 7-6. Entering a named destination - Dialog Box

The dialog box will suggest a good guess for the named destination. If the next word found is a number, it assumes that a chapter is to be set and adds the “Ch” prefix to the suggestion. If first detected word is non-numeric, the dialog suggests a *Sect*-prefix for “Section”. However you shall always follow the naming conventions as described in 7.1.4.1 “Step 1: Choose the right destination name” on page 7-88.

> This function will automatically add the “M8.newlink.” prefix to the entered named destination (see next paragraph).

Using the special Plug-In function is highly recommended because it corrects the positioning information. Details are described in 12.1.2.1 “Position correction” on page 12-185.

Alternative Step 4: Set the destination with prefix “M8.newlink.”

If “The ND.API plug-in” is not installed, then open the named destination window according to 7.1.3 and use the  button to create a named destination. An entry will open

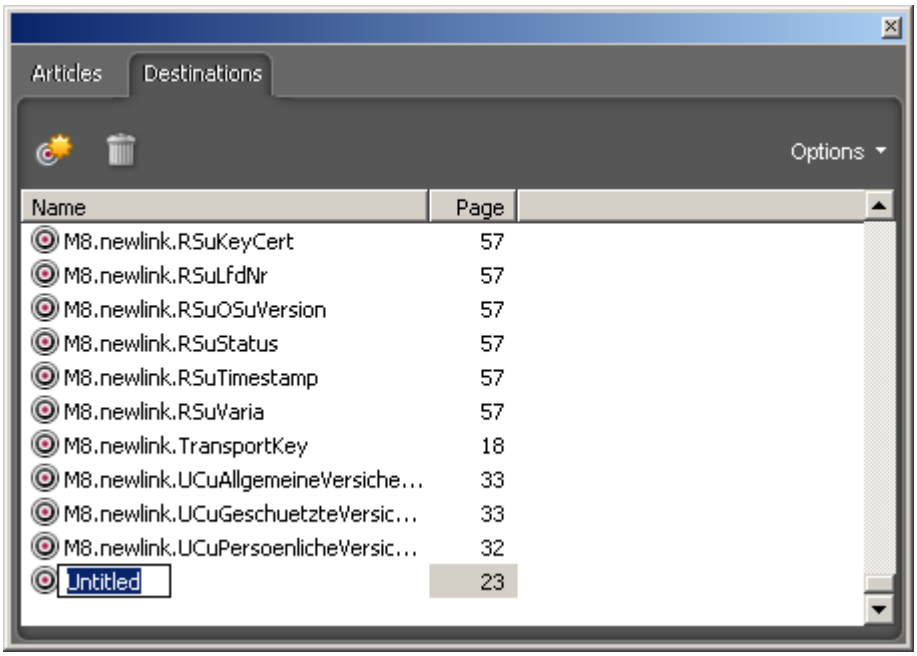


Figure 7-7. Entering a named destination

Enter with prefix: *M8.newlink.MyDest* to add a named destination *MyDest*. You can leave the window open or close it, the named destination is set after you confirmed with “Enter”.

The M8.newlink. notation is compliant with Adobe FrameMaker when an explicit (manual) named destination is set. It comes very handy for ezRead since the added destinations are easily recognizable from automatically generated destinations in other editors (e.g. "Bookmarks" in MS-WORD.

7.1.4.5 Step 5: Save the PDF

After having entered all your named destinations, save the PDF file. Only then the named destinations are stored in the PDF.

Note: Do not use the *SaveAs* function until you have converted an MS-WORD file generated document as described in 8.4 “Converting MS-WORD links” on page 8-132.

Documents created by *Microsoft Word* will perhaps loose some links (e.g. those of the table of contents) because with the *SaveAs* function, Adobe runs a *cleanup process*. As MicroSoft Word does not set stabile connections, Adobe Acrobat assumes that those loose links can be washed. A full list of MS-WORD artifacts is given in 8.3 “Unwanted artifacts in MS-WORD PDF conversion” on page 8-128.

7.1.5 How to create named destinations automatically

Creating named destinations can be done automatically. How exactly this is possible depends on the information available in a PDF file. Some some tools are available to help. In the following we discuss different cases, starting with the “best of all” situations and developing into the worst condition.

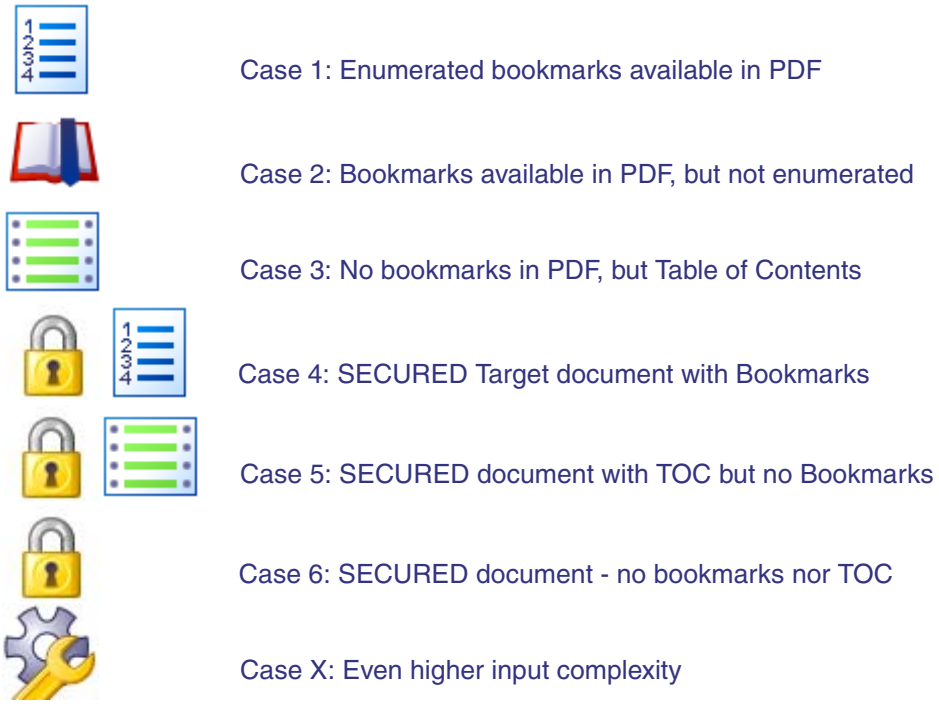


Figure 7-8. Dispatching the different cases

These cases cover the most typical situations with external documents. The case “SECURE document with non-enumerated bookmarks” remains to be added on-demand.

7.1.5.1 Case 1: Enumerated bookmarks available in PDF

This is the best case. Bookmarks are available in the bookmark navigation panel.

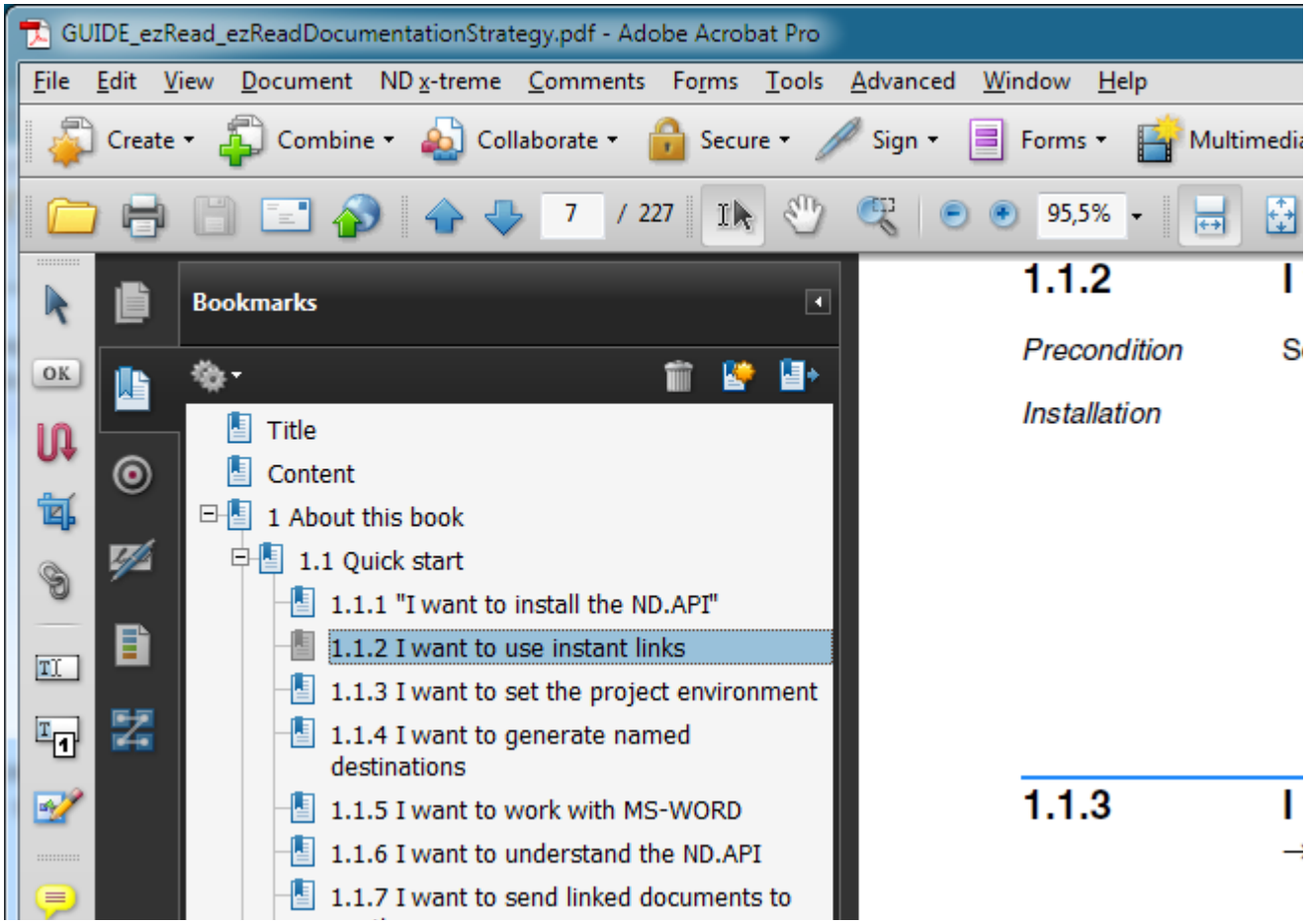


Figure 7-9. Bookmarks in a PDF file

Note: If the bookmark navigation panel is not available you can select it by *View - Navigation Panels - Bookmarks*.

The ND.API plug-in can then use this bookmark information in order to create named destinations. It is important, that the bookmarks are *already enumerated*, because The ND.API plug-in will generate names for the destinations *from the enumeration*. *Chapter 4.2.7* becomes *Ch4p2p7*. (For the naming conventions, see 7.1.4.1above).

To create named destinations from bookmarks simply use

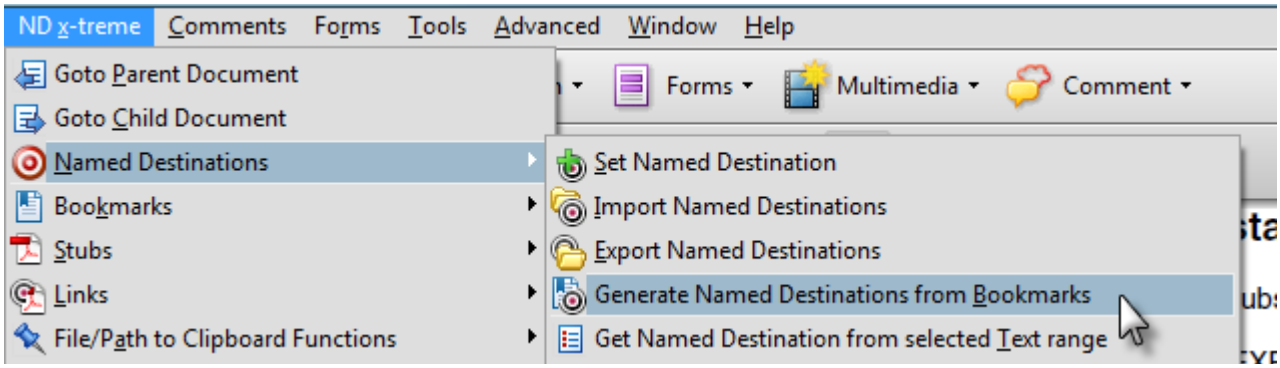


Figure 7-10. Generate named destinations from bookmarks

The named destination box will then contain the generated named destinations including the *M8.newlink* prefix.

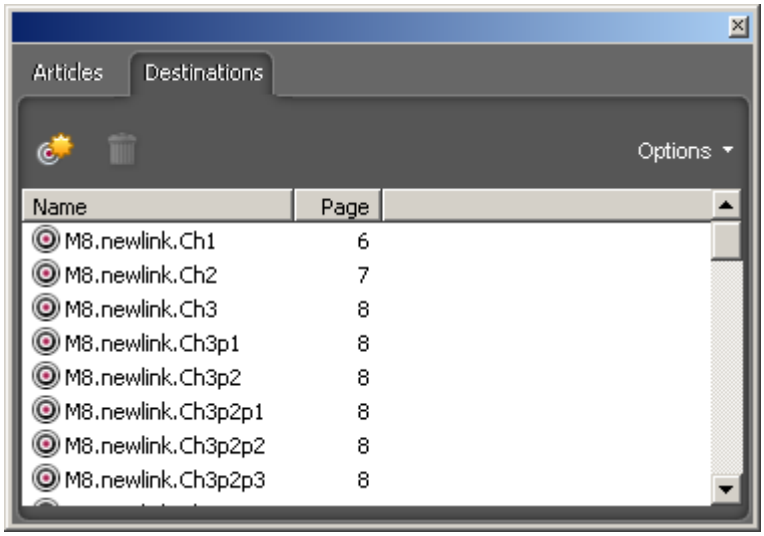


Figure 7-11. Generated destinations

7.1.5.2 Case 2: Bookmarks available in PDF, but not enumerated

You first need to enumerate the bookmarks in the file. How to enumerate bookmarks can be read in chapter 7.1.8 below.

Enumeration is necessary to allow the The ND.API plug-in the generation of reasonable names for the destinations.

After the enumeration you can proceed according to 7.1.4 “How to create a named destination manually” above.

7.1.5.3 Case 3: No bookmarks in PDF, but Table of Contents

Unfortunately, many documents do not have bookmarks available because on PDF conversion the author forgot to enable the bookmark generation in the PDF settings. It also happens, that stupid bookmarks are generated, if the PDF conversion settings have not been done correctly when a PDF was created from a WORD document.

If you have the source document (e.g. MS-WORD) the best idea is to convert it again to PDF using the correct PDF conversion settings. Experiments, however, have shown that even then it can be very difficult sometimes to get the bookmarks generated.

If the PDF contains a table of contents, then another process can be done. The ND.API plug-in is able to import Bookmarks from a CSV file.

This CSV file can be generated from an extract of the PDF’s table of contents using the The following source is a basic hotkey script useable to realize the Instant Jump to Stub function.

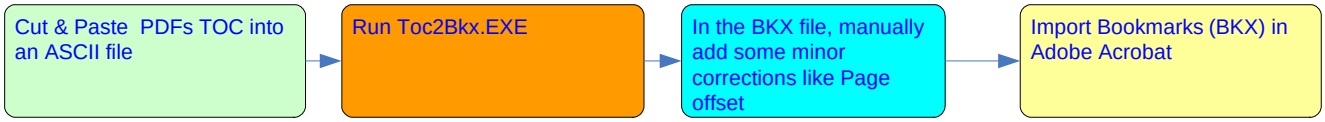


Figure 7-12. Import Bookmarks from Table of Contents

Step 1: Cut out TOC: go to the table of contents of the PDF file and mark the actual text after selecting the  key. The marked text might look as follows

1 About this book 1-5

2 ezRead Documentation Architecture 2-7
2.1 Requirements for documentation excellence 2-7
2.2 ezRead Rationale and Document Strategy Overview 2-9
2.3 Questions and answers 2-13

3 ezRead Reader's Guide 3-17
3.1 Book organization 3-17
3.2 Product documentation 3-17
3.3 Accompanying documentation 3-23
3.4 Project documentation 3-24
3.5 Tips and hints for reading ezRead 3-25

Figure 7-13. Mark the table of contents

Step 2: Paste the content **:** into an ASCII file which can be created with any ASCII editor (UltraEdit, CodeWright, Windows NotePad).

The ASCII file *MyDoc.Txt*. in the above example would look like

```
1 About this book . . . . . 1-5
2 ezRead Documentation Architecture . . . . . 2-7
2.1 Requirements for documentation excellence . . . . . 2-7
2.2 ezRead Rationale and Document Strategy Overview . . 2-9
2.3 Questions and answers . . . . . 2-13
3 ezRead Reader's Guide . . . . . 3-17
3.1 Book organization . . . . . 3-17
3.2 Product documentation . . . . . 3-17
3.3 Accompanying documentation . . . . . 3-23
3.4 Project documentation . . . . . 3-24
3.5 Tips and hints for reading ezRead . . . . . 3-25
```

Warning: Sometimes, a TOC-line is *broken* up into two lines. You have to repair these situations because the Toc2Bkx.EXE does not detect the broken line. Every line in the ASCII file shall finally end with a page number.

Step 3: Repeat... If you have a large table of contents, spanning over more than one page, you might have to repeat the above cut-and-paste procedure to collect the entire table of contents in the ASCII file. PDF files do not always allow to pick data spanning over pages, or they do at least add some undesired other information (e.g. page numbers and footnotes) in.

Step 4: Save to file: *Save* the ASCII file as *MyDoc.Txt*.

Step 5: Apply Toc2Bkx.EXE: In a command line window run Toc2Bkx.EXE. We assume, that this program is available and the path where it resides is part of the PATH environment variable. This all will be managed by the installation procedure according to 11.7 "Set/Clear documentation environment" on page 11-178.

```
F:\MyDir>toc2bkx MyDoc.TXT
```

will generate ***MyDoc.BKX*** which looks like

```
MyDoc,MyDoc.pdf,0
M8.newlink.Ch1,5,XYZ,0:0:0,1,"1 About this book"
M8.newlink.Ch2,7,XYZ,0:0:0,1,"2 ezRead Documentation Architecture"
M8.newlink.Ch2p1,7,XYZ,0:0:0,2,"2.1 Requirements for documentation excellence"
M8.newlink.Ch2p2,9,XYZ,0:0:0,2,"2.2 ezRead Rationale and Document Overview"
M8.newlink.Ch2p3,13,XYZ,0:0:0,2,"2.3 Questions and answers"
```

```
M8.newlink.Ch3,17,XYZ,0:0:0,1,"3 ezRead Reader's Guide"
M8.newlink.Ch3p1,17,XYZ,0:0:0,2,"3.1 Book organization"
M8.newlink.Ch3p2,17,XYZ,0:0:0,2,"3.2 Product documentation"
M8.newlink.Ch3p3,23,XYZ,0:0:0,2,"3.3 Accompanying documentation"
M8.newlink.Ch3p4,24,XYZ,0:0:0,2,"3.4 Project documentation"
M8.newlink.Ch3p5,25,XYZ,0:0:0,2,"3.5 Tips and hints for reading ezRead"
```

Step 6: Adjust pages: The *first line* of *MyDoc.BKX* has a special meaning.

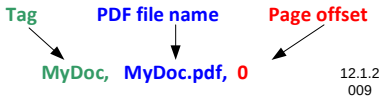


Figure 7-14. Import Bookmarks from Table of Contents

The **Tag** contains an *identifier* for the document. Although it does not technically hurt, for use with other tools (e.g. Auto-Documentation Tools) this identifier should be *unique* throughout the entire project.

The **PDF file name** contains the file name of the file from which the bookmarks were exported. If on import, the current file name does not match this field, a warning will be given out by The ND.API plug-in asking the user for confirmation.

The **Page offset** field is important. Some books start numbering with roman letters (i, ii, iii) and continue with numeric restart some pages later (1.2.3...).

Change the **Page offset** field in *MyDoc.BKX* to the number of pages *preceding* the first '1'-numbered page.

Step 7: Import the bookmarks into your document: Like shown in Figure 7-10 import the *MyDoc.BKX* into the *MyDoc.PDF* file using *Advanced - Manage Destinations - Import Bookmarks*.

The ND.API plug-in import function has a smart *title search feature* which searches the bookmark text on the designated page and fine-tunes the position to the exact position of the chapter.

Disabling title search feature

The header search feature can be switched off *setting the Z-component* of the X:Y:Z triple to a *non-zero value*.

Then continue with the "Case 1: Enumerated bookmarks available in PDF" scenario.

Note: Sometimes the entries in a table of contents are ALL UPPERCASE whereas the related headings in the text are just normal text. Therefore the comparison on importing bookmarks (→ 12.1.10 "Export/Import Bookmarks" on page 12-194) is *case-insensitive* and even corrects the CAPITALIZED bookmark entries of the *Table of Contents* to their actual appearance in the text.

7.1.5.4 Case 4: SECURED Target document with Bookmarks

It often occurs, that a document is SECURED. Then it cannot be changed, neither is it possible to add bookmarks or named destinations. Unfortunately this often happens on the most important external documentation like standards or data sheets.

A document which has no named destinations normally cannot be referred to by chapter numbers. As you cannot add named destinations to a SECURED document (neither generated them from bookmarks) this seems to be a problem.

The solution to this problem is Stub management. Through the The ND.API plug-init is possible to open an ASCII file with the fake extension *PDF*, that contains information to the actual PDF file including page and position information (→ Linking FrameMaker to SECURED documents). This fake PDF file is called a ”stub” - it is very small and serves as anchor into the SECURED target file.

Opening a stub will invoke the associated Adobe Product (Acrobat Reader) and finally open that document, that the stub refers to. As the position information is contained in the stub, the Acrobat Reader (with the The ND.API plug-in installed) will use that information to open the actual file at the desired position.

The stub technology is explained in “Linking FrameMaker to SECURED documents”. However, you need to generate the stubs first. This can be done by the “Generate named destinations from bookmarks” function

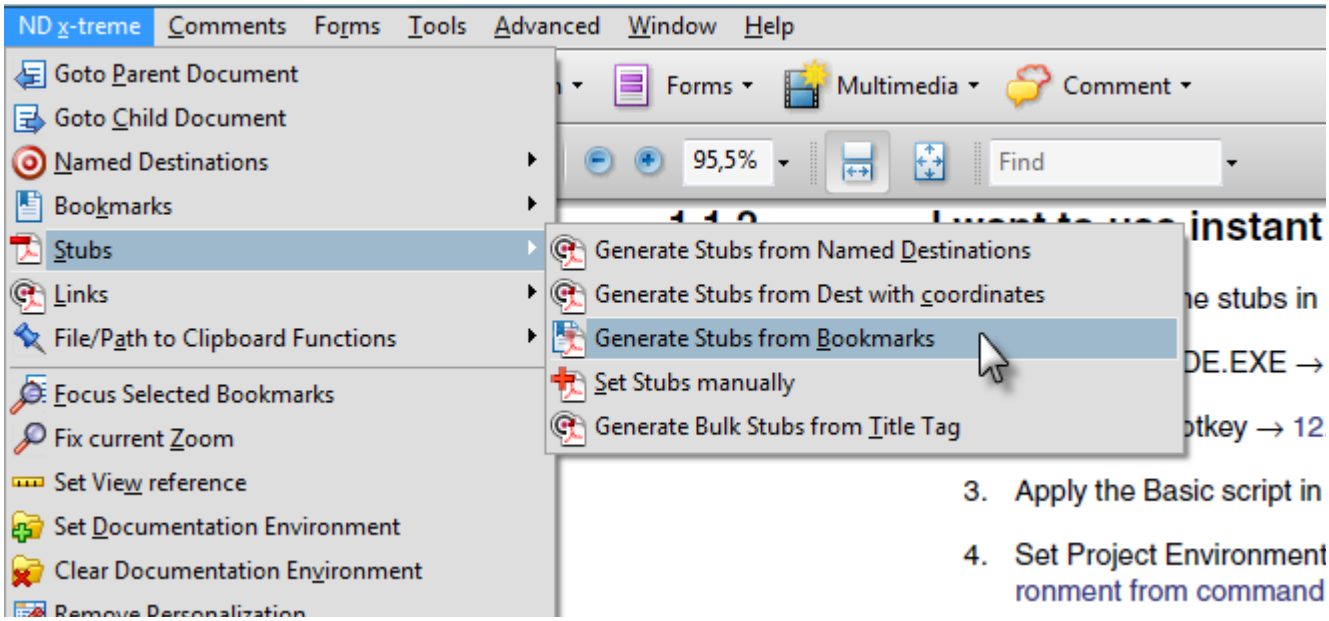


Figure 7-15. Generate stubs from bookmarks

A dialog box will appear to prompt you for a unique identifier of the target document.

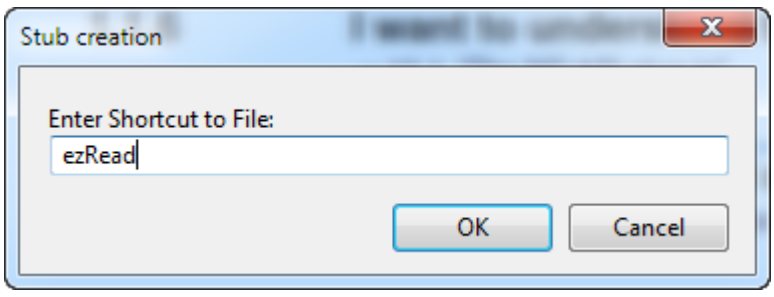


Figure 7-16. Entering a Document Identifier

Enter a unique identifier for your document. It is a good idea to organize those identifier for the entire project's relevant documentation - also helpful for many other instances that refer to documents.

The name should be short to make it easier for references to read. As a reference, *ASY_Ch3p2* reads better then *SystemArchitectureSpecification_Ch3p2*. The technology is explained in 9.3.3 "[ezRead#9.3.3] notation and syntax" on page 9-145.

On "OK" the function will generate a stub file for every bookmark found in the document. As that can be many files per document, the stubs are all organized in a particular *STB directory* which is part of the environment (→ 11.7 "Set/Clear documentation environment" on page 11-178).

A reference into the SECURED document should be implemented as a reference to its corresponding stub.- using the Document <Document ID>_<NamedDest> notation. (Example: Spec3_Ch7p3p1). → Stub management.

The stub acts like a *Named Destination*, but is externally located in the STB directory. The targets can easily be tested using the Explorer and clicking on those stubs in the STB directory.

> Since the number of files in the STB directory may easily run to thousands of (very small) files, it is not a good idea to find a stub file by scrolling through the STB directory. We highly recommend to type the file name in - since the naming convention is generic, typing is by far the mos efficient way to enter a reference.

7.1.5.5 Case 5: SECURED document with TOC but no Bookmarks

Bookmarks cannot be created in a SECURED document. Again the Stub management will be used.

Step 1 : First generate a bookmark index file (BKX) as described in "Case 3: No bookmarks in PDF, but Table of Contents".

Once the have created the TOC file and converted the text with the Toc2Bkx.EXE you will have a bookmark index file (BKX → 12.1.10.1 "BKX:Bookmark CSV File structure" on page 12-194).

Unfortunately the TOC of a document does not provide the proper y-Position, only pages. The ND.API plug-in repairs this situation automatically when bookmarks are imported (→ 12.1.10 "Export/Import Bookmarks" on page 12-194). However, in a SECURED document bookmarks cannot be imported.

Step 2: Import the Bookmark file. - As expected, after import a dialog box will warn, that the bookmarks could not be imported – this message comes from Acrobat and cannot be supressed.

Figure 7-17. Warning on SECURED files

However, an underlaying mechanism of the ND.API *updated the BKX file* to contain the correct positions.

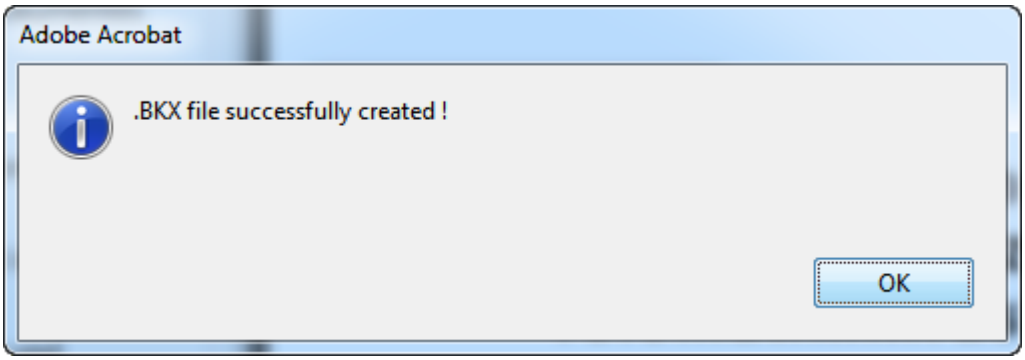


Figure 7-18. BKX created message

Step 3: Use Bkx2Stb.EXE in order to generate stubs from the (corrected) bookmark file. Through the correction during the (intentionally unsuccessful) bookmark import the stubs will point to exact chapter positions where the ND.API did find them. You can always add manual stubs later in the project time.

7.1.5.6 Case 6: SECURED document - no bookmarks nor TOC

This is by far the worst case. If no Table of Contents is available, the Toc2Bkx.EXE (→ "Case 3: No bookmarks in PDF, but Table of Contents") cannot be used to retrieve bookmark information. In this rare case, the stubs need to be entered manually.

Use the "Set Bookmark Stubs" function

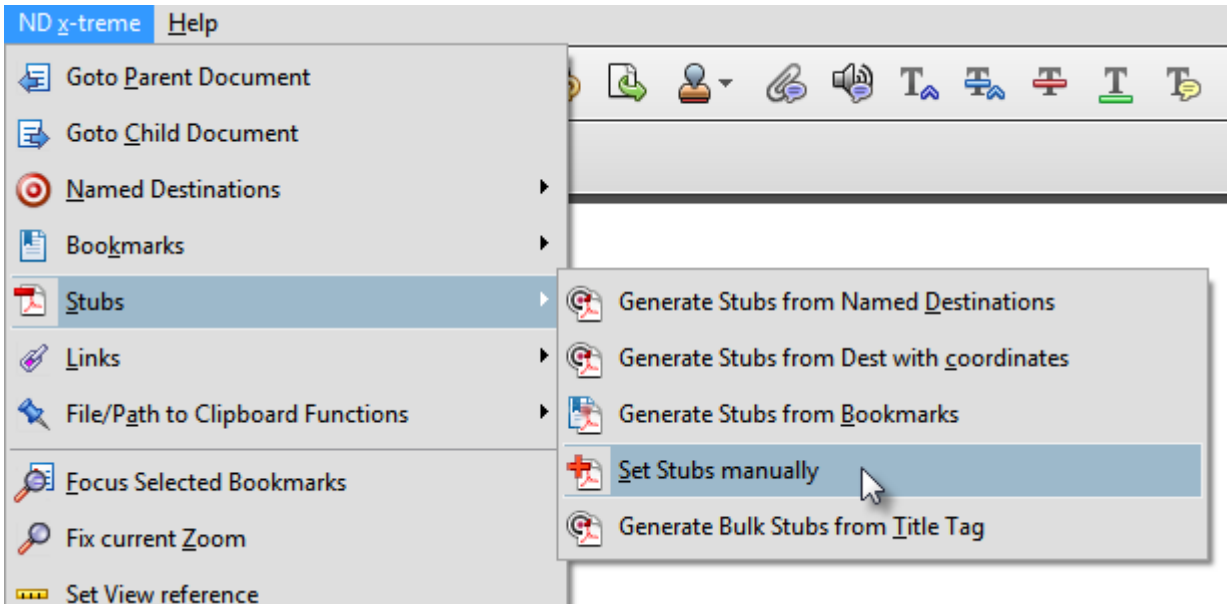


Figure 7-19. Setting stubs manually

Only once a dialog box will appear to prompt you for a unique identifier of the target document.

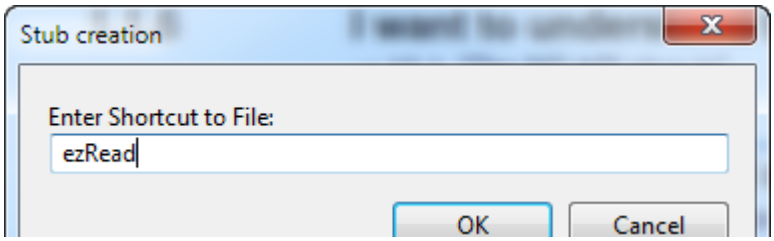


Figure 7-20. Entering a Document Identifier

The Document ID is maintained until you ever press "*Cancel*" in this or the following dialog box. Another dialog box appears

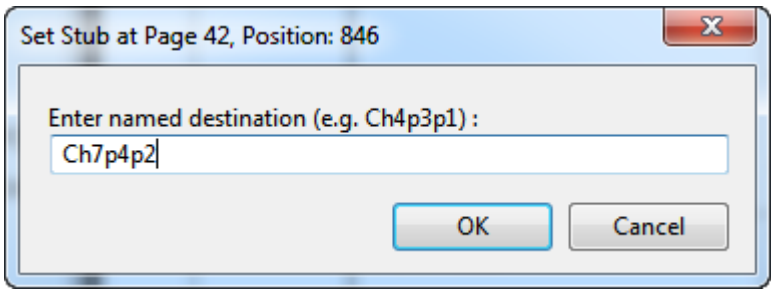


Figure 7-21. Enter named destination for stub

The naming convention for any generated stub is similar to what is described in "Step 1: Choose the right destination name".

A single stub file will be generated in the STB directory of the project environment. The stubs are all organized in a particular *STB directory* which is part of the environment (→ 11.7 "Set/Clear documentation environment" on page 11-178).

7.1.5.7 Case X: Even higher input complexity

It can be even more complex to add bookmarks automatically. If a PDF has odd page numbering (e.g Roman letters: ixx) or appendices that restart page numbering, auto-generation can become a good piece of work.

Some tricks on the way to success shall be mentioned here but will not be exploited in detail.

Extract Pages	Split the document into separate parts using the "Extract Pages" menu. The workout the book marking scheme for those single parts and after you did, rejoin the parts
Tune BKX file	The file generated by Toc2Bkx.EXE can be manually tuned with an ASCII editor (ULTRAEDIT32, CodeWright, NotePad).
Combine BKX import file	you may import different <Bookmarks.BKX> in order to form the entire list.
Manual entry	In smaller documents, it can even be affordable to enter the bookmarks manually. Having this done, you should export the bookmarks. That can be very helpful on later versions

7.1.6 Stub technique

Through the The ND.API plug-in it is possible to open an ASCII file with the extension *STB*, that contains information to the actual PDF file including page and position information. This STB file is called a "*stub*" - it is very small and serves as anchor into the SECURED target file.

7.1.6.1 Understanding stubs

Opening a stub will invoke the associated Adobe Product (Acrobat Reader) and finally open that document, that the stub refers to. As the position information is contained in the stub, the Acrobat Reader (with the ND.API plug-in installed) will use that information to open the actual file at the desired position.

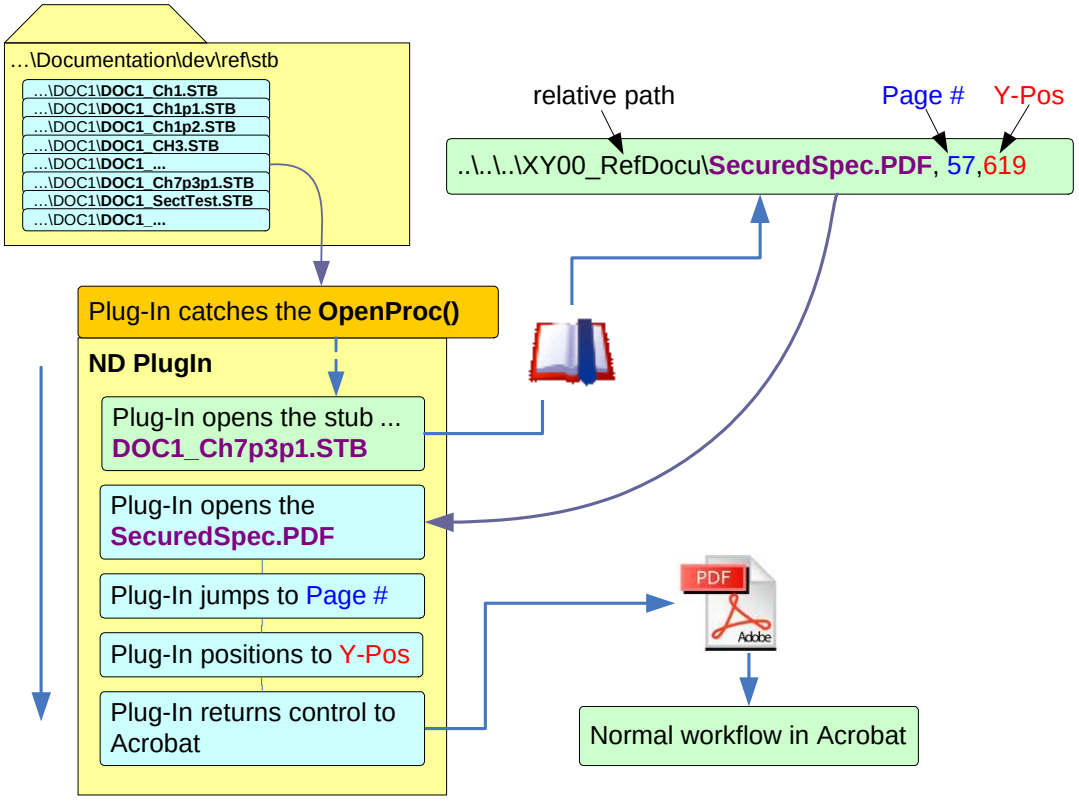


Figure 7-22. Architecture of stub anchor mechanism

The figure above shows how the ND.API catches the *OpenProc( )* of the Acrobat Reader and interpretes the filename/content of the stub in order to find the correct position in the target document.

7.1.6.2 When to use stubs

It is an excellent idea to use stubs whenever you write a document (e.g. MS-Word) that will be printed as PDF. The advantage is, that the target document may change place and name but by regeneration of the stubs (→ 12.1.14 "Generating Stubs from named destinations") all links – even in a PDF file will resolve correctly into the new target.

The only reason where you would not like to link via stubs would be when you export the documention tree to a customer who does not have and want to install the ezRead environment. Therefore the export function (→ 12.1.26 "Exporting linked files") removes all stub reference and replaces them by direct links to PDF with the exception of SECURED files where it is not possible to link directly.

Using a central repository for the stubs will also allow Instant Jump to Stub which is a very powerful technique to refer to chapters of PDF even without having to establish a cross reference at all.

7.1.7 Content based destination generation

Very often it occurs that the content of PDF files contains tables with identifiers. For instance, there are tools which generate lists (e.g. DOORS tool) and it might be very helpful to address the items of such lists. The following table is an example of a typical generated list that contains identifiers of interest for anchor points.

Table 7-3. Example for a requirements list – Req IDs shall be marked

Req ID	Description	Tag
AE531	The product shall work 24/7	high
AE538	The availability shall be 99,998%	high
AE682	.....	....

On the implementation a software developer might want to generate references to the fulfilment of the above requirements. Therefore it would be desirable if on every ReqID (in the example above) a named destination could automatically be placed at every ReqID.

The appropriate method is called Content based destination generation.

The ND.API plug-in provides a special function to generate automated content based named destinations.

> This feature is only available from **Acrobat X** since from version X it is possible to select ranges in contrast to text. An extension for prior versions could be provided, but this will only be implemented on demand.

Mark the filter area: Using the marker function in Acrobat X  drag the mouse over that area of a representative page that you would like to filter for tags.

Note: Always mark the entire possible range where you might find relevant tags (here AEx). If the table on your selected page does not span from top to bottom you should even

then mark the entire range *until the bottom of the page* because otherwise tags outside your marked area won't be found.

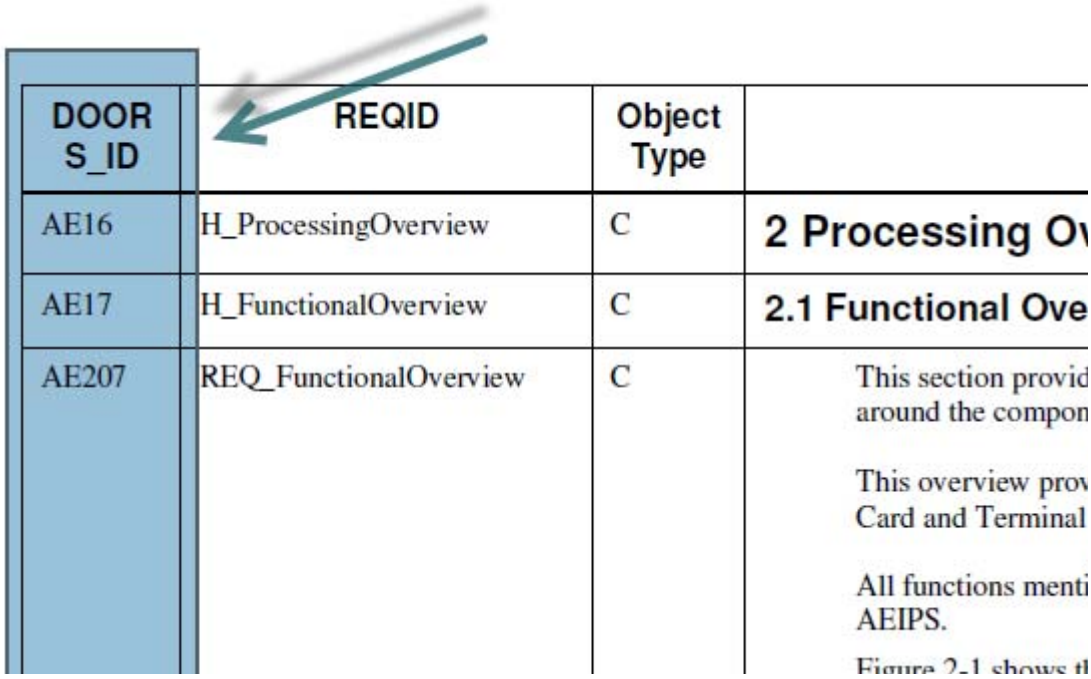


Figure 7-23. Marking the filter area

Select the conversion function: Use the **Named Destinations – Get Named Destination from selected Text range** function to specify your search criteria.

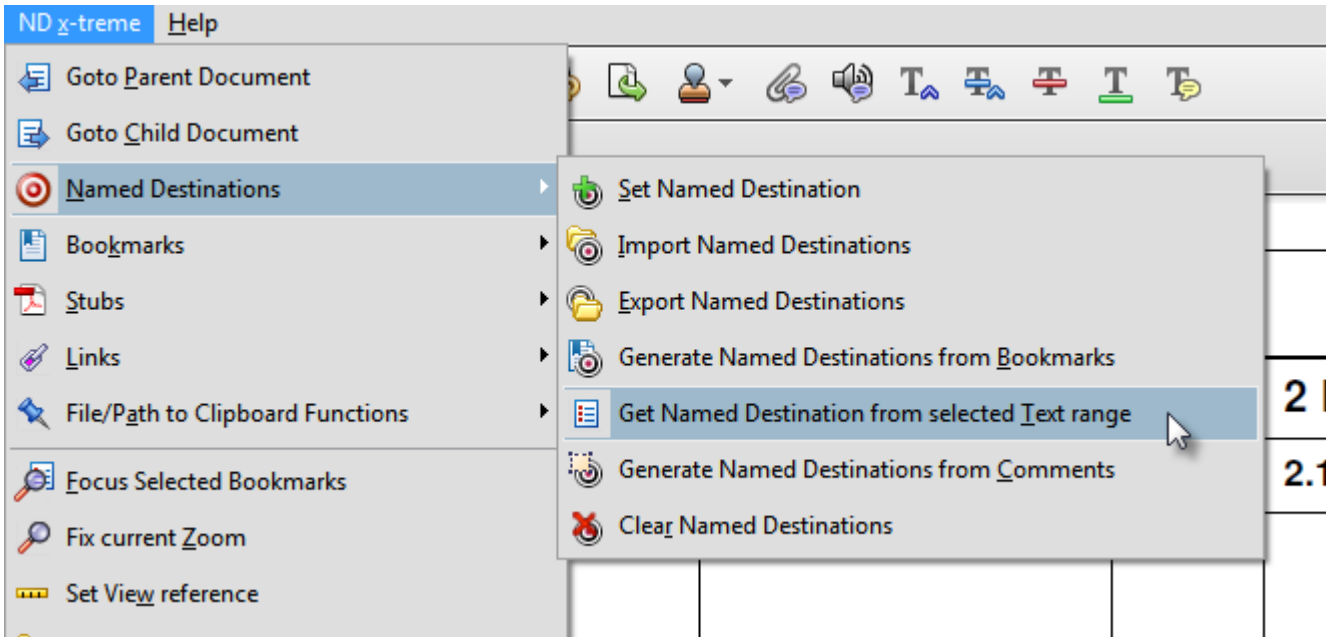


Figure 7-24. Select the conversion function

Set text filter: A dialog box appears which prompts you for the input of a regular expression:

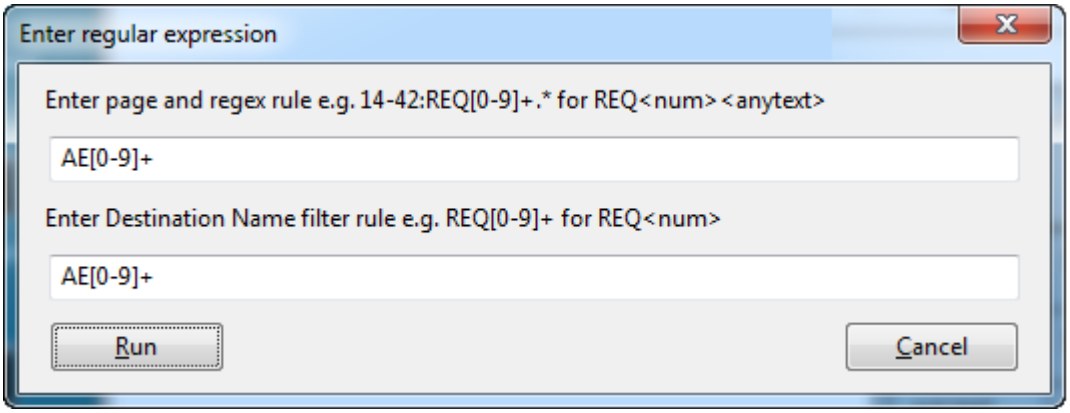


Figure 7-25. Enter regular expression

Regular expressions are explained in 7.3 “Regular expressions”. The above entry mandates an "AE" somewhere in the string followed by one (+) or more (+) occurrencies of a numeric [0-9] value.

The second field "Enter Destination Name filter rule" is **optional** but it can be come very handy to filter the actual name from the result of the first field's evaluation. There are situations where you need to be more tolerant in the regex rule to catch all your desired targets. You might have to enter a tolerant search criteria but you do not want to have the results to become the actual name of the named destinations.

Therefore you may filter the first filter's output (= potential destination names) through the second filter which extracs the actual destination name from the found expression.

Note: Leave the 2nd field empty if you do not need a special name extraction.

Note: Without explicit page specification the creation of named destinations will start from the page where you placed the filter until the end of the document..

Specifying pages: A special syntax - not complying to regular expressions - is available to explicitly select the pages to be processed.

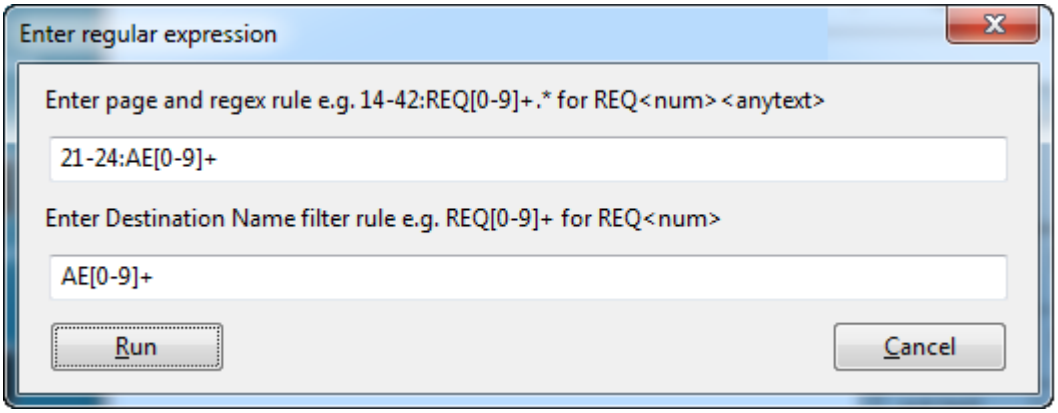


Figure 7-26. Selecting pages – here from page 21 to 24

Place the page range or a single page ahead of the regular expression and put a colon as separator. In Figure 7-26 the example shows a page selection from 21 to 24. The colon is mandatory to inform the ND.API that the first section is not part of the regular expression.

The final result will generate named destinations, if the targets were numbers then the "Ch" (Chapter) prefix is added in order to avoid numbers only as destination trailer.

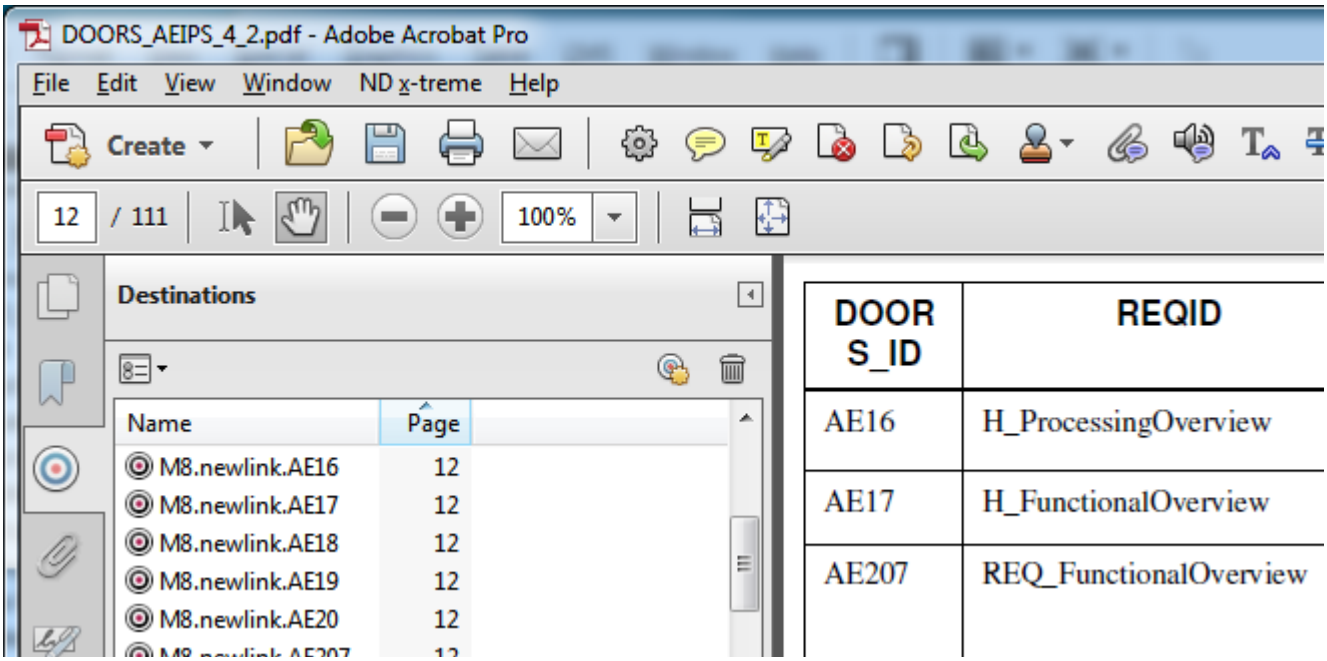


Figure 7-27. Final result

7.1.8 How to enumerate bookmarks

If a source document (PDF) does not contain enumerated bookmarks, you might want to add enumeration later. The ND.API plug-in offers an appropriate function.

7.1.8.1 Enumerate-it

An Un-enumerated bookmark panel may look as follows

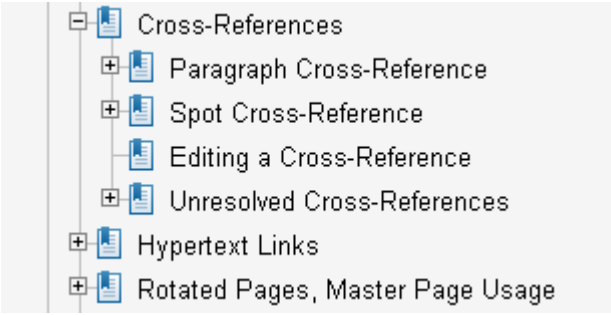
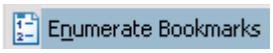


Figure 7-28. No enumeration in bookmarks.

Use the *Bookmarks – Enumerate Bookmarks* function from the Acrobat Menu



land the bookmarks get enumerated

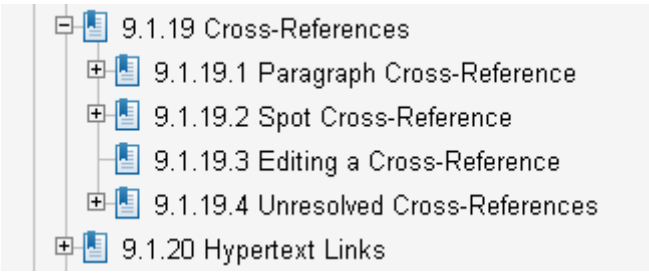


Figure 7-29. Auto enumerated bookmarks

7.1.8.2 Special cases

MS-WORD has quite a strange attitude towards the generation of bookmarks. While the *chapters* in a WORD document are actually enumerated, the generated bookmarks are *not*.

Using the Enumerate-it function helps, however, one thing has to be respected. The bookmark enumeration should of course *match the definitions* in the actual chapters. A bookmark like ***7**.1.2 - FAQs* should not link to a chapter ***6**.1.2 - FAQs*.

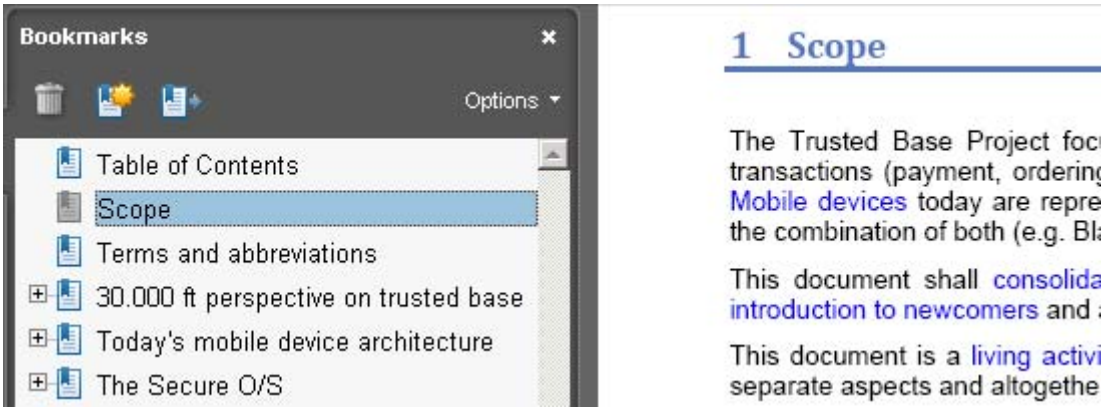


Figure 7-30. Chapter 1 (Scope) starts with second bookmark

Figure 7-30 shows an example where the “Scope” is numbered as Chapter One (“1”) but its relevant bookmark in the PDF file is in the second place - preceded by the “Table of Contents” mark.

Such *Chapter-Bookmark mismatch* can happen if bookmarks are taken into the auto-enumeration, that are not actually enumerated in the *text*. For MS-WORD this does often occur for the table of contents. In particular in external documents it is not possible to ask an editor to correct his/her settings in the source.

Auto-enumeration cannot know about this shift and will consequently start its enumeration with the first bookmark entry. But that would mean, that the “Scope” would get the wrong enumeration.

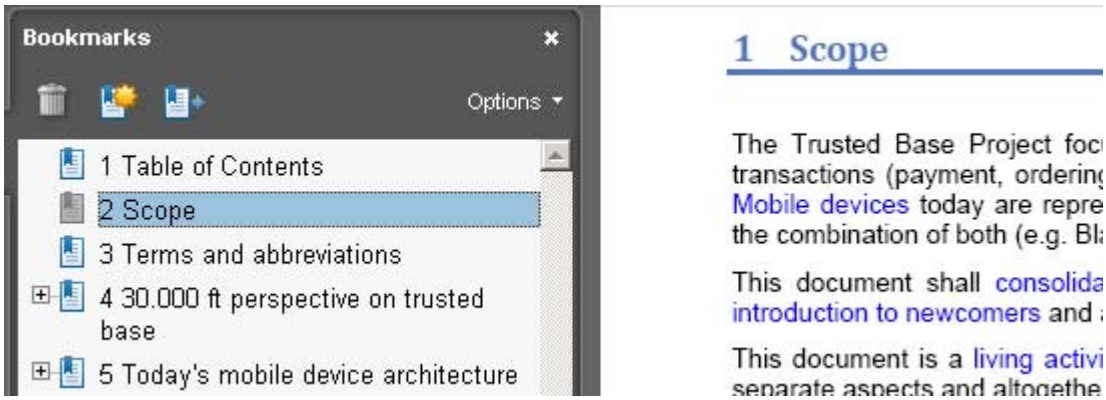


Figure 7-31. “Scope” badly enumerated because enumeration took first bookmark wrongly

This situation however, can be easily fixed.

- **Select the bookmark** to be enumerated first.
- **Bookmarks - Enumerate Bookmarks** will then take the selected bookmark as first bookmark to enumerated and start from there.
- Alternatively you can also select a *range of bookmarks* with “Click / Shift-Click” or a selection of *particular* bookmarks with “Click / Ctrl-Click”. On enumeration only those selected bookmarks will be taken into enumeration.

7.1.9 Enumerate – Merge function

The recent release of the ND.API function has extended the enumeration to an even more intelligent algorithm. A document may already have enumeration. While in previous versions, the enumeration was not possible on top of an enumerated document, now the enumeration will search the first numbered bookmark and continue the numbering found on this occurrence by the correct numbers to further chapters. Also the chapters prior to the first found numbered bookmark will be numbered accordingly.

The lowest enumeration level possible is "1"

The function will be automatically synchronize bookmarks, if you start enumeration **without any bookmark selection**.

7.1.10 How to jump to a named destination

This is well described in “Hypertext Links” in volume 2 page 6-11. and in the FrameMaker User Manual. A special variant to personalize the position of the destination alignment (typically at the top of the window) is explained in 12.1.23 “Set ND view reference” on page 12-207.

For use with MS-Word refer to 8.1.3 “Link to external PDF with named destination”.

7.1.11 Linking FrameMaker to SECURED documents

Target documents to be referenced may be *SECURED*. This means, *it is not possible* to add named destinations so those documents. We do, however, recommend the Stub management as presented in Chapters 7.1.5.4 to 7.1.5.5.

If you do not want to use Stub management, FrameMaker, allows to open a document at a particular page number using the “Special - Hyperlink” with “Open Document at Page Number.”

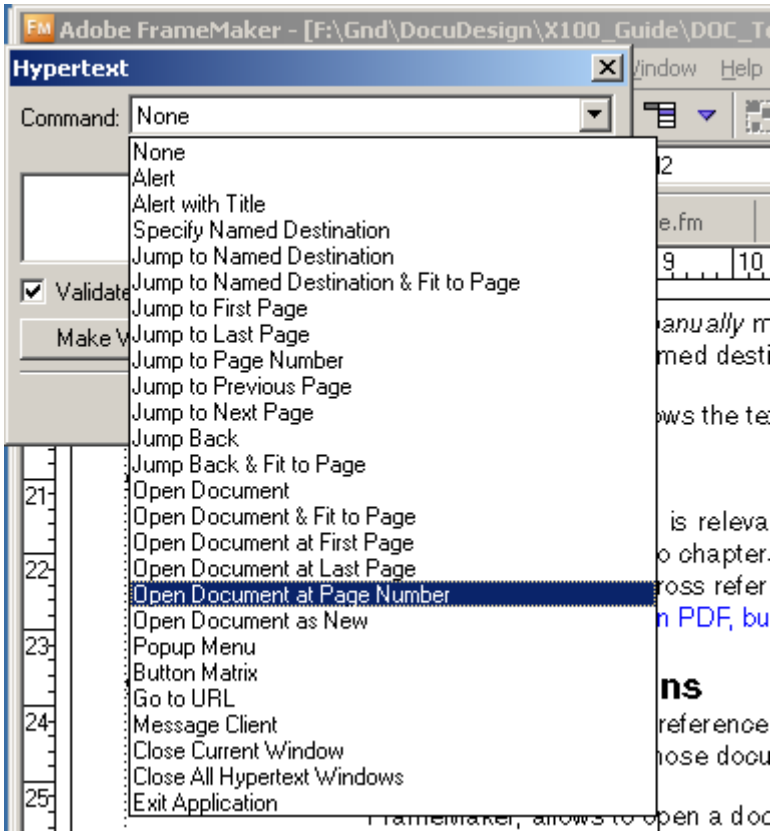


Figure 7-32. Open a document at an explicit page number

Fixing link info in PDF files

Unfortunately on generation of a PDF file, FrameMaker creates *links to named destinations* that look like “P.n” e.g. “P.3” stands for *page 3*. The target document, however, does typically not have neither these named destinations nor any other one. And as we could not add named destinations to SECURED documents - the problem remains.

Wrong “PDF” extension on stub files

Working with Stub management throws another flavour of the problem. When referring to external (stub) files, FrameMaker always converts the extensions of a referenced file into the extension “*.PDF*”. While this is an excellent feature in most cases, for Stub management it is not wanted because we would loose the relation to the stub files with “*.STB*” extension.

Fixing FrameMaker artifacts

The ND.API plug-in provides the function “Convert PDF to STB (FrameMaker)” which automatically translates page links into their explicit equivalents. After processing, those links will work into the target pages without using named destinations. The same function “Convert PDF to STB (FrameMaker)” also visits links to stub files and repairs the extension back from “*.PDF*” → “*.STB*”.

Such a conversion needs to be done *once* on any document created with FrameMaker. The effort is very small and therefore affordable.

7.1.12 Instant Jump to Stub

A very powerful feature as a consequnce of Stub management is the instant link feature.

In association with the 02_AutoHotkey_L_Install.exe a script may be used that allows to open a PDF link instantly right from the current editor. This editor may be an ASCII editor, PowerPoint, EXCEL, MS-WORD or any other tool. The Instant Jump to Stub technique requires a particular link notation

Any text written with a notation as [ISO4#5.3] will allow to jump immediately to a chapter in a document that created stubs with the prefix "ISO4". To make this, do the following steps

1. **Select** any portion of your text that includes the desired link

   .... with a notation as [ISO4#5.3] will allow to jump immediately ....

2. Press the "instant link" function key – using the recommended script Basic HotKey script – 03_Basic.ahk use the F12 key.

3. The script will copy your selection to the clipboard and strip the [...] content. Then the script will dispatch the notation *ISO4#5.4* by translating it to a *stubs filename* as in the following example

   ```
   ISO4#5.3 → ISO4_Ch5p3.stb
   ```

4. The file found from the initial notation will be launched and the link will be resolved into the associated chapter (here 5.3).

The elegance of this function is a very quick link. The mechanism does not establish any link into the edited document, this is always subject of the syntax of the present editor – however the function may be used even in documents that do not support hyperlink jumps at all e.g. XML editors.

7.1.13 Instant Jump with dialog box

The above function can also be used to directly open a file with known DocID. The Basic HotKey script – 03_Basic.ahk provides the key combination **Ctrl-Shift-Alt F12** in order to open a dialog box to enter the unique code.

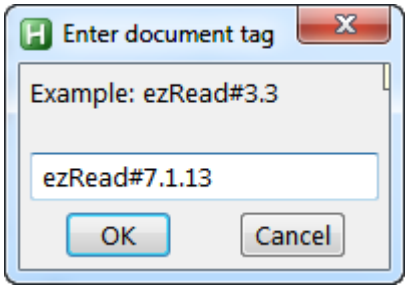


Figure 7-33. Open a document through a dialog box

7.2 Exporting books and references

According to the preceding chapter, the final books are recommended to be stored in the X000_Books directory.

Sending book trees to 3rd party

Very often, one or more of the architecture books shall be sent to a *third party*, being a customer or another development site (e.g. subcontractor). The internal directory structure as suggested in 6.4 “Where to put documents” on page 6-70 is efficient for the development but can be hard to accept by a receiver. Instead it does require a very simple “message” on where to store documents.

Therefore the The ND.API plug-in provides an export function which copies all files that belong to an inter-linked cluster into two directories.

7.2.1 Using the ND.API Export function

Use the “Export Linked Files” function

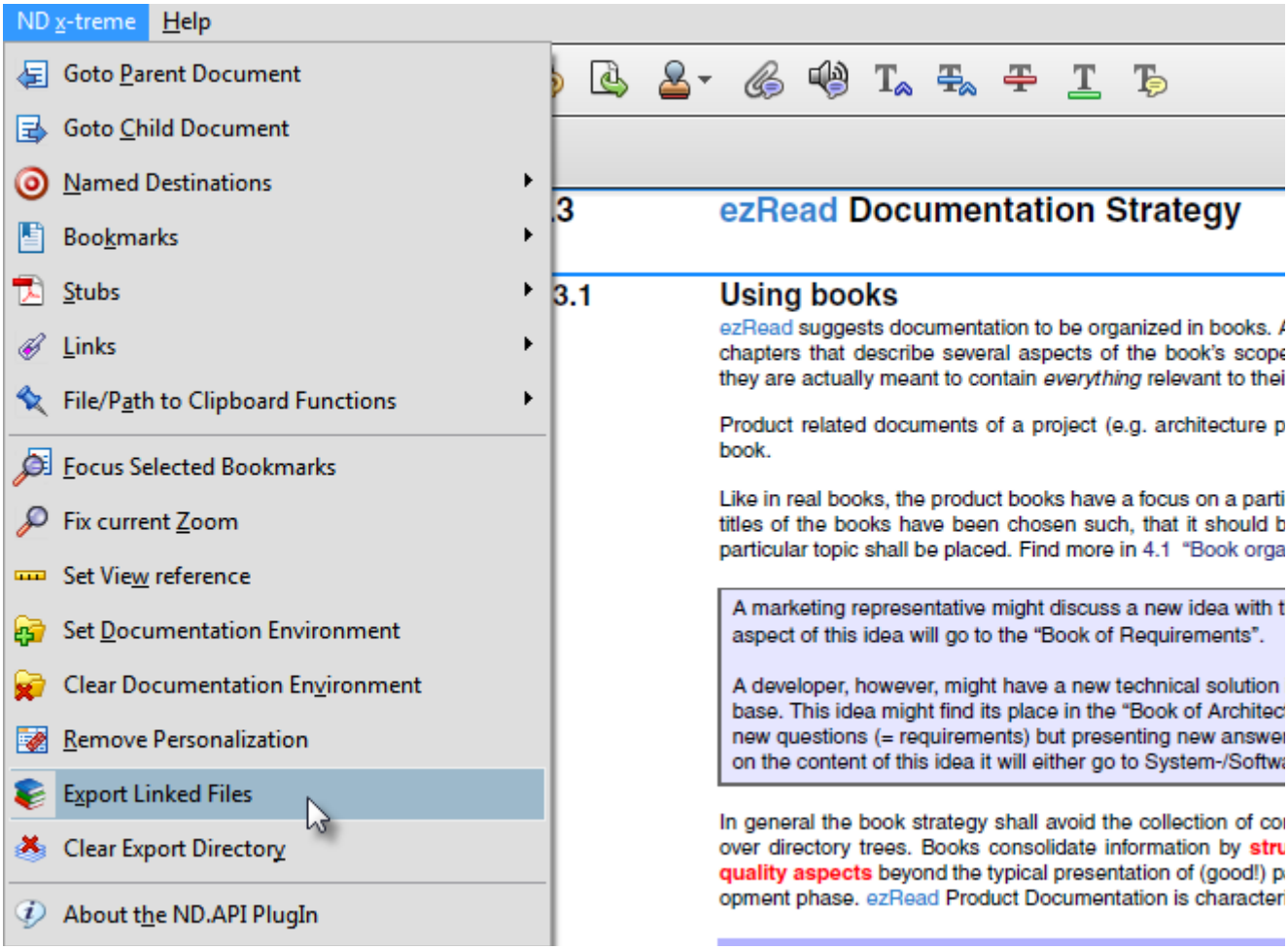


Figure 7-34. Exporting linked files

the The ND.API plug-in will copy all files of a document tree (starting with the open document) into three Target directories of an "export directory" as shown in Figure 7-35. The links in every linked document and its children will be converted during this copy to relatively point into the new Target directories.

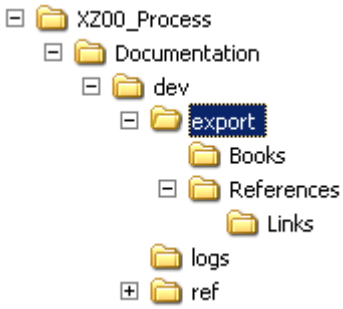


Figure 7-35. Export directory in Process Tree

Since the number of documents - when visiting any child of the child, can be large, a dialog opens asking for the number of sub-levels to export.

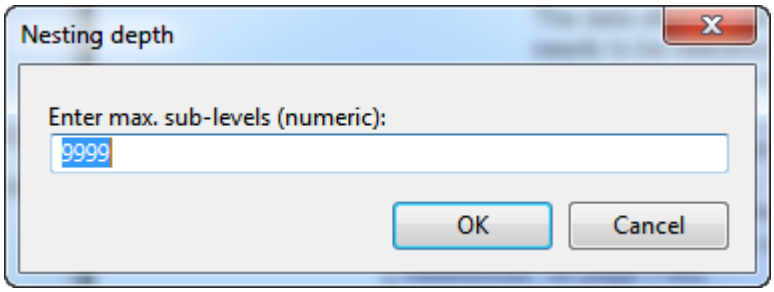


Figure 7-36. Entering nesting depth

- Level 0 will export only the present document
- Level 1 will export only documents being linked by the present documents but will not visit the links of those
- Level 2 .... N will visit as many child levels as you have entered. On large numbers (e.g. 9999) the final tree will be complete (except for missing links) but also take more space and time to process.

Depending on your customer's needs it may be appropriate to export the entire tree or only a restricted number of documents.

7.2.2 Target directories

Three target directories will be maintained for export

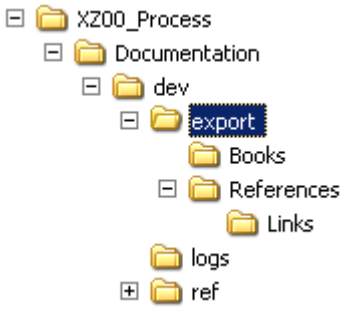


Figure 7-37. Target directories under "export"

Books. holds the PDF files that are the entry points, typically those books from which the *Export Linked Files* function was started.

References. holds any document referred from of the link tree, regardless where it came from, even other drive's file will have this target.

References\Links

holds the STB anchors, whenever any document of the link tree makes use of the Stub management, which is necessary when working with Linking FrameMaker to SECURED documents. The stubs are modified to point to the new directory. Their targets will copied and updated, if possible. Not only Linking FrameMaker to SECURED documents will use stubs for referencing, e.g. if a MS-WORD document cannot establish the named destinations, it might also use Stub management to address named destinations.

When processed, all stubs are replaced by direct links. This is always possible unless a link into Linking FrameMaker to SECURED documents are found. Of course there a named destination cannot be applied and stubs need to remain and shall be part of the delivered package.

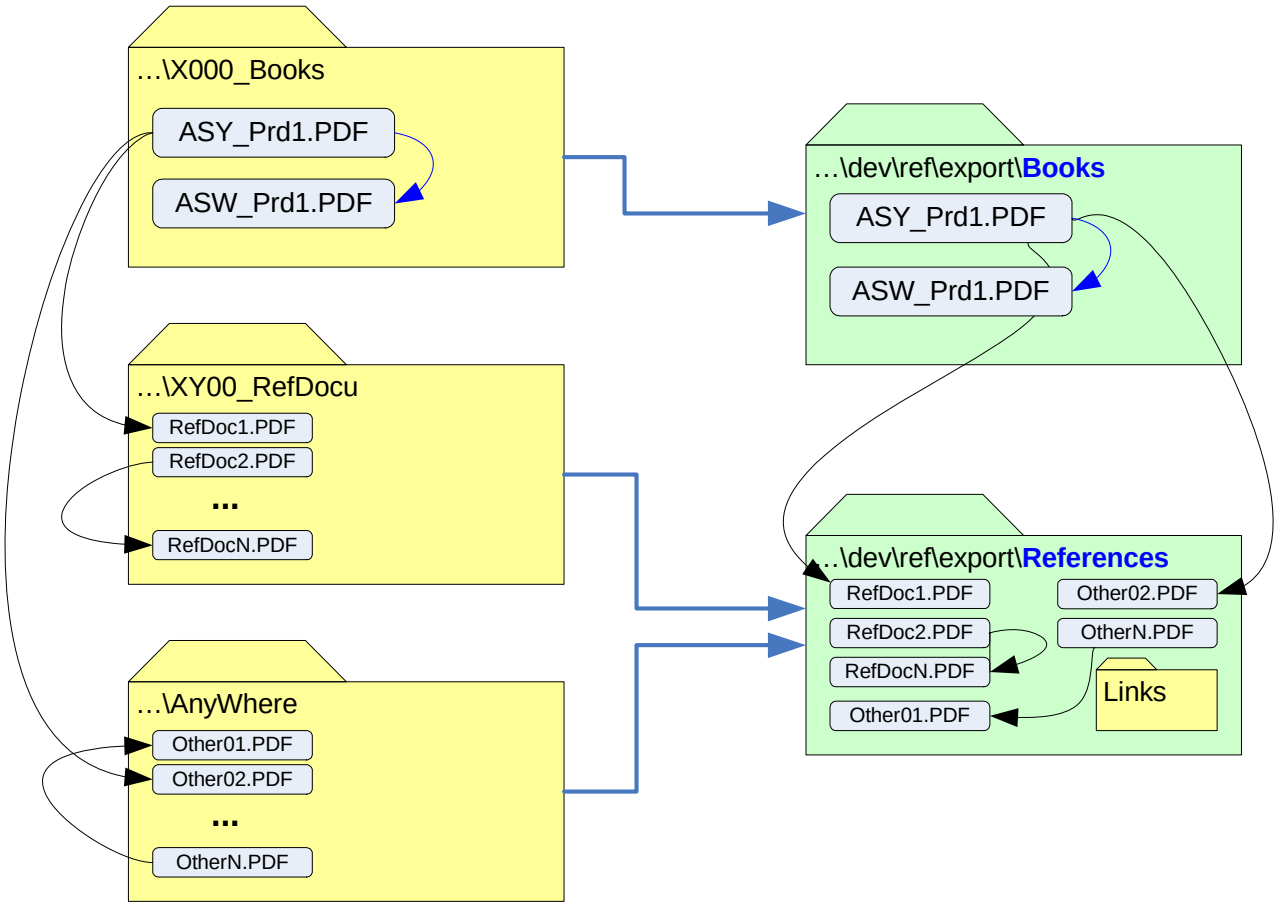


*Figure 7-38. Export from working directory into **Books** and **References**.*

Other directory names on export

The new target directories give up the specific path names X___ as suggested in 6.4 “Where to put documents” on page 6-70 but apply **BOOKS** and **REFERENCES** as easy-to-remember and acceptable directory names.

The **BOOKS** directory will store only the root book which is the starting point whereas the **REFERENCES** directory holds any other file being referred from the main book or any referenced book.

References are changed

The copy operation is smart i.e. while a document is copied into its new directory its references will be changed such that it applies correctly from the new target directories (Books/References). The operation is a recursive cyclic-safe process - therefore any linked document will be found and copied and changed.

Receiver shall maintain export directories

The Books/References tree structure, however, must be kept by the receiver. Due to the simple structure, the receiver is likely to accept this scheme.

On demand, it is possible to create another *export mechanism*, that runs into one directory only. That, however, would have to be paid with less intuitive entry points. (**BOOKS** directory).

7.2.3 References from Books to Books

If a Book **A** references to another book **B**, then **B** shall be placed into the *Books* directory (as being a book). The question is, how can **B** be detected as a book and not as any other reference file?

How to recognize a book

The ND.API plug-in scrutinizes the *Document Properties* of the Adobe file and recognizes whether the file **B** was created from a ".BOOK" FrameMaker file. If this is found true, then **B** is recognized as BOOK file and placed in the Books directory.

> The ND.API plug-in is searching the phrase "BOOK" in the Title-property of the document."BOOK" may be written anycase "book" or "bOoK" would all be valid findings as the search is not case sensitive.

By this mechanism, FrameMaker BOOKS will automatically be placed in the *...\Books* directory, all other documents will go to the *...\References*.

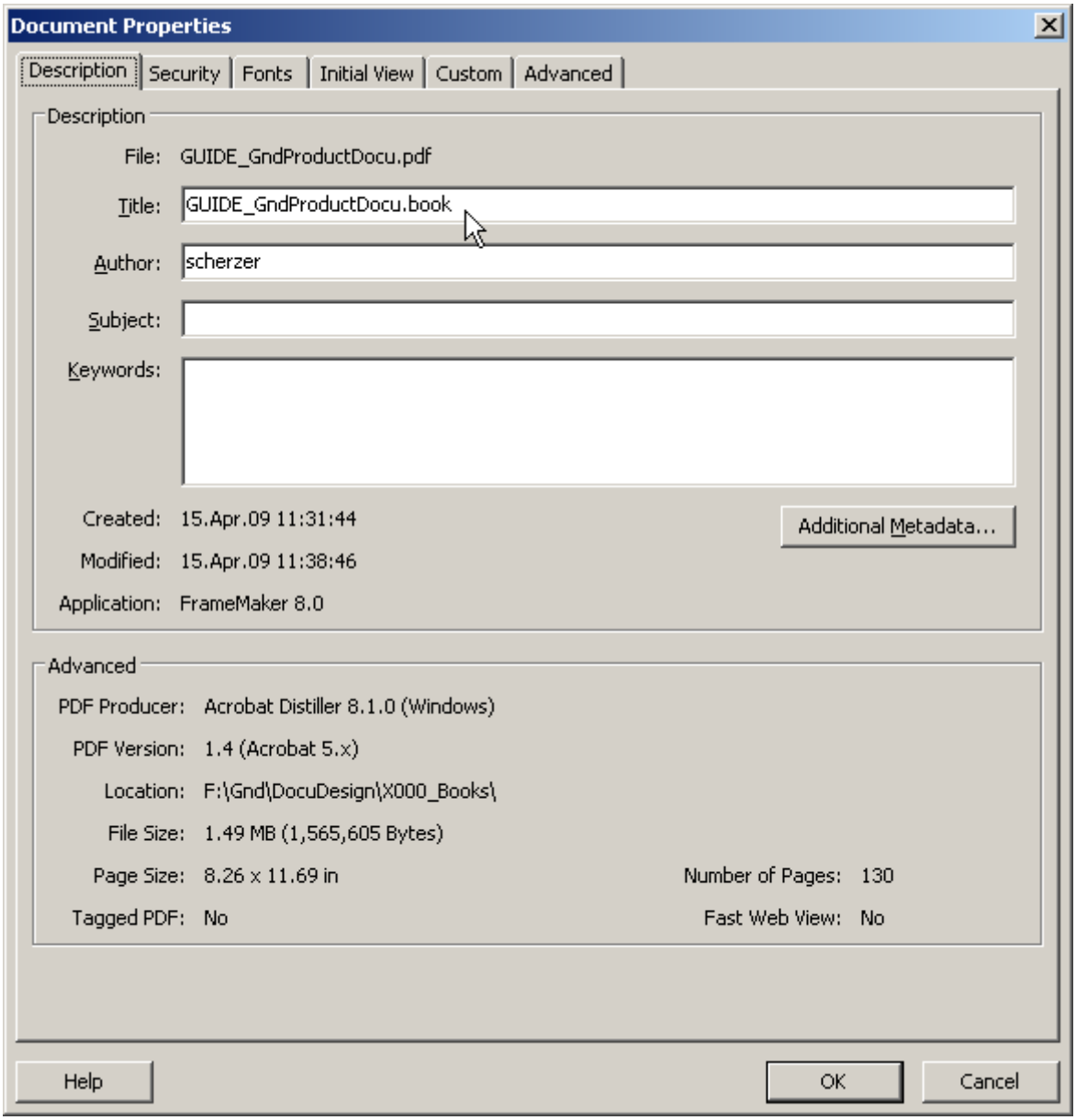


Figure 7-39. Detect a book from the PDF properties : Title

Manually make a target to a 'Book'

If a user wants any other document to go to the BOOKS directory for any reason, s(he) can alter the title in the Acrobat Reader with "File-Properties[Description]"entry in the appropriate file (**B**) containing anywhere in the title description the word "BOOK". The other way around - removing the "BOOK" phrase from the title works accordingly.

7.2.4 Missing references

The ND.API plug-in will note the user, if files are referred, but not present in the tree. A <bookname>.TXT log file will be created to list the missing files. The page number will also be displayed

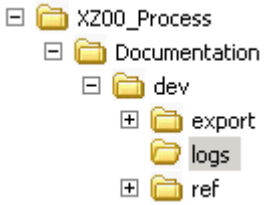


Figure 7-40. Logs are stored in the logs-directory

Sample record

An example of one record in the log file could show like

```
ASY_Cams Page 185  Missing: F:\camsH\X000_FmDocu\X200_ASY\ASY_CV_CCA.pdf
```

From the PD page number, the editor may derive the missing link.

> Exporting files can be very practical to check whether there are missing links in a document tree.

7.2.5 Clearing the export directory

The export directory can be cleared. Use the “Clear Export Directory” function.

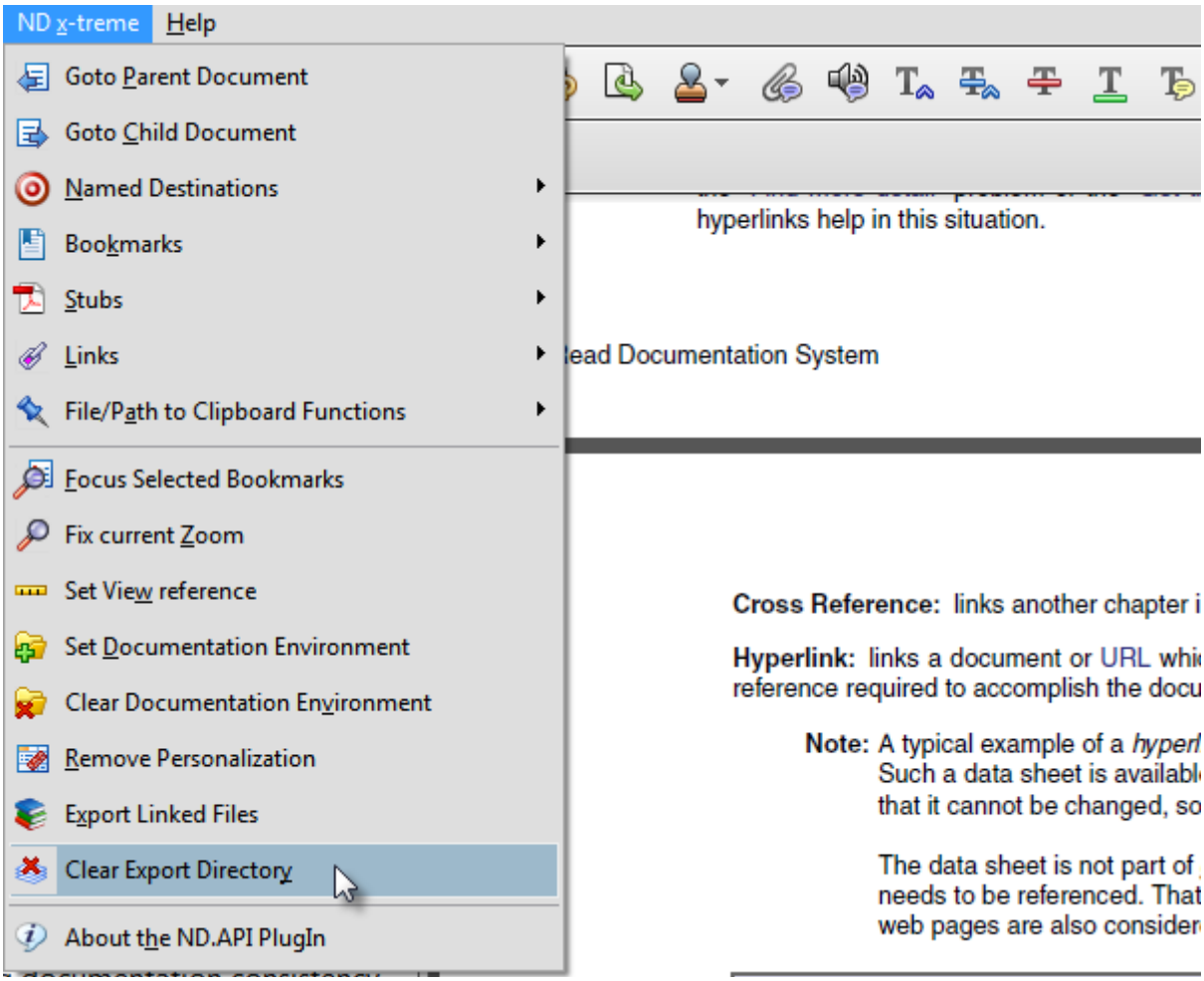


Figure 7-41. Exporting linked files

This will remove all files from the export directory. Typically you can export more than one book. The files in the export directory *will not be cleared automatically* on another export because it may be the user’s intention use to create an export package with several books.

The Clear export directory function allows a full cleanup of the export directory including the removal of the BOOKS and the REFERENCES directory. The EXPORT directory itself will not be removed.

7.3 Regular expressions

The following text as well as the coding was taken from RESearch.cxx – Regular expression search library by Ozan S. Yigit (oz), Dept. of Computer Science, York University.

The cases [ ] explain the constructs that can be used for the evaluation of regular expressions in the ND.API.

[1] *char* matches itself, unless it is a special character (metachar): . \ [ ] * + ^ $ and ( )

[2] . the dot (.) matches any character.

[3] \ the backslash (\ ) matches the character following it, except:

- \a, \b, \f, \n, \r, \t, \v match the corresponding C escape char, respectively BEL, BS, FF, LF, CR, TAB and VT;

  Note: Note **\r** and **\n** are never matched because the parser searches are made line per line

- if not in posix mode, when followed by a left or right round bracket (see [7]);
- when followed by a digit 1 to 9 (see [8]);
- when followed by a left or right angle bracket (see [9]);
- when followed by d, D, s, S, w or W (see [10]);
- when followed by x and two hexa digits (see [11].

Backslash is used as an escape character for all other meta-characters, and itself.

[4] *[set]* matches one of the characters in the set.

- If the first character in the set is "**^**", it matches the characters **NOT** in the set, i.e. complements the set. A shorthand S-E (start dash end) is used to specify a set of characters S up to E, inclusive. S and E must be characters, otherwise the dash is taken literally (eg. in expression [\d-a]).

The special characters "]" and "-" have no special meaning if they appear as the first chars in the set. To include both, put minus (**-**) first: **[-]A-Z]** or just backslash them.

[-]|] matches these 3 chars,

[]-|] matches from] to | chars

[a-z] any lowercase alpha

[^-]] any char except - and]

[^A-Z] any char except uppercase alpha

[a-zA-Z] any alpha

[5] * any regular expression form [1] to [4] (except [7], [8] and [9] forms of [3]), followed by closure char (*) matches **zero** or more matches of that form.

[6] + same as [5], except it matches **one** or more. Both [5] and [6] are greedy (they match as much as possible).

[7] a regular expression in the form [1] to [12], enclosed as \(form\) (or (form) with posix flag) matches what form matches. The enclosure creates a set of tags, used for [8] and for pattern substitution. The tagged forms are numbered starting from 1.

[8] *\n* a backslash (****) followed by a digit 1 to 9 matches whatever a previously tagged regular expression ([7]) matched.

[9] \< a regular expression starting with a \< construct \> and/or ending with a \> construct, restricts the pattern matching to the beginning of a word, and/or the end of a word. A word is defined to be a character string beginning and/or ending with the characters A-Z a-z 0-9 and _.

Scintilla extends this definitionby user setting. The word must also be preceded and/or followed by any character outside those mentioned.

[10] *\l* a backslash followed by d, D, s, S, w or W, becomes a character class (both inside and outside sets []).

d: decimal digits

D: any char except decimal digits

s: whitespace (space, \t \n \r \f \v)

S: any char except whitespace (see above)

w: alphanumeric & underscore (changed by user setting)

W: any char except alphanumeric & underscore (see above)

[11] *\xHH* a backslash followed by x and two hexa digits, becomes the character whose Ascii code is equal to these digits. If not followed by two digits, it is 'x' char itself.

[12] a composite regular expression xy where x and y are in the form [1] to [11] matches the longest match of x followed by a match for y.

[13] *^* a regular expression starting with a **^** character **$** and/or ending with a **$** character, restricts the pattern matching to the beginning of the line, or the end of line. [anchors] Elsewhere in the pattern, **^** and **$** are treated as ordinary characters.

8 Working with MicroSoft Office (MS-WORD)

8.1 Linking files with MS-WORD 120
8.1.1 Linking any external document 120
8.1.2 Link to external MS-WORD document 120
8.1.3 Link to external PDF with named destination 121
8.1.4 Link to external MS-WORD document with DOC bookmark 122
8.2 Printing MS-WORD document as PDF 124
8.2.1 Using Bookmarks from MS-WORD 126
8.3 Unwanted artifacts in MS-WORD PDF conversion 128
8.3.1 DOC file links are not converted to PDF targets 128
8.3.2 MS-WORD creates URLs as links 128
8.3.3 MS-Word produces unstable links 130
8.3.4 Office 2003 does not take in bookmarks straight forward 130
8.3.5 Office 2003 does not allow straight ND 131
8.4 Converting MS-WORD links 132
8.5 Use case - from DOC2DOC to PDF2PDF 133
8.5.1 1. Write target with bookmarks 134
8.5.2 2. Converting to PDF with WORD bookmarks 134
8.5.3 3. Generate Named Destinations from Bookmarks 137
8.5.4 4. Generate Stubs 137
8.5.5 5. Link Stubs 138
8.5.6 6. Converting source file to PDF 138
8.5.7 7. Convert MS-links 138
8.6 Creating bookmarks from comments 138

We highly recommend to use *Adobe FrameMaker* to make "Using books" easy. MS-WORD does not allow the creation of books and has several other known and reported disadvantages. (e.g. writing documents with more than 500 pages was reported to make MS-WORD instable. Also the comfort of adding named destinations is poor. With OFFICE2010 and later, MicroSoft did improve this aspect and today (in particular through the introduction of the XML based DOCX format) you can write a large book in WORD withouth to fear that it breaks on printing / PDF production.

MS-WORD will often be used as the default editor in companies because Adobe FrameMaker is rather expensive. On PDF creation, MS-WORD generates PDF files but the link features are established poorly, even in OFFICE2010. A list of the artifacts is explained in 8.3 below.Therefore "The ND.API plug-in" has a special MS-WORD conversion filter - acting on the produced PDF files - that corrects all the artifacts.

MS-WORD allows to link to *external files* through the "Hyperlink" function.The main sections of this chapter describe aspects to be covered when using MS-WORD.

- "Linking files with MS-WORD" explains how documents are linked from MS-WORD.
- "Printing MS-WORD document as PDF" explains how to print MS-WORD documents such that most of the cross reference information is take into the PDF file. This, however does not work as there are "Unwanted artifacts in MS-WORD PDF conversion" which are explained the third chapter.
- "Converting MS-WORD links" shows how to cope with the above artifacts - finally making may impossible things of the past work.
- An important use case is described in "Use case - from DOC2DOC to PDF2PDF".
- Finally there is a possibility to generate very special target files by using comments for the creation of PDF bookmarks and named destinations This is explained in "Creating bookmarks from comments"

8.1 Linking files with MS-WORD

8.1.1 Linking any external document

To link any file, *select the area* that should react to a link

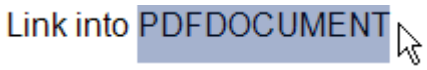


Figure 8-1. Linking an external PDF file from MS-WORD

Selecting "**Insert - Hyperlink" (Ctrl-K)**" will bring up the following file selection dialog.

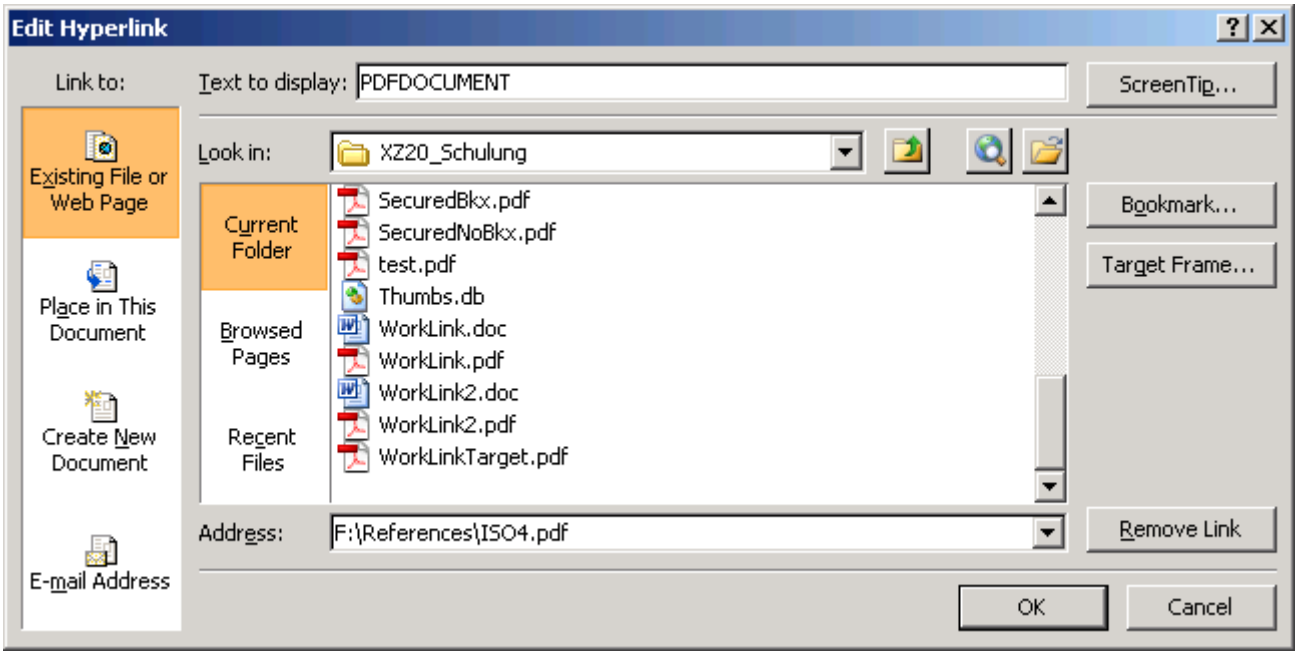


Figure 8-2. Linking an external PDF file from MS-WORD

Select the target file (*here ISO4.PDF*) to create the link and press "OK". The link is established and shows like follows

Figure 8-3. Link established in MS-WORD.

8.1.2 Link to external MS-WORD document

This is very much the same as in the preceding paragraph except that the linked document is a MS-WORD document.

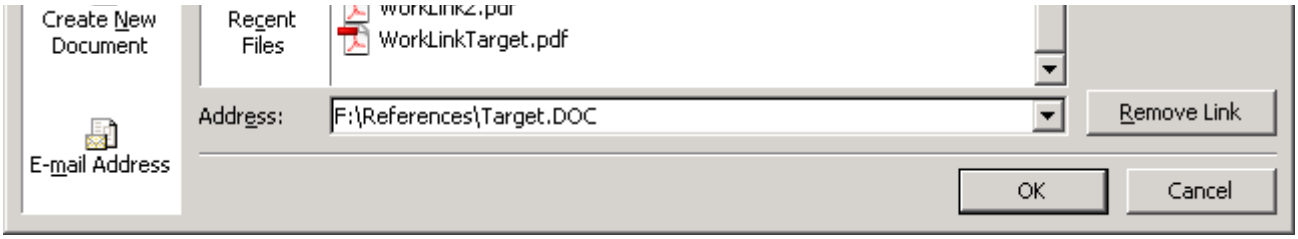


Figure 8-4. Linking an external MS-WORD file (.DOC)

No PDF target on printing

In contrast to FrameMaker, MS-WORD does not translate a link into a MS-WORD document into a link to the associated PDF document, when producing a PDF from the source. Once this is understandable, because the evidence of the conversion of the linked document into PDF is not necessarily given.

On the other hand they create a PDF with links into a MS-WORD document which is a media break that is not recommend. The solution to this is the "Convert URI to stub" function in the ND.API. With the conversion in the PDF file, the link will *automatically be altered* to the target file's *PDF representation* - in the above example the link would be set to "Target.*PDF*".

8.1.3 Link to external PDF with named destination

Linking a named destination in a target document will be done as follows. In the File-Open Dialog, a #-sign shall be appended to the file name, followed by the actual named destination

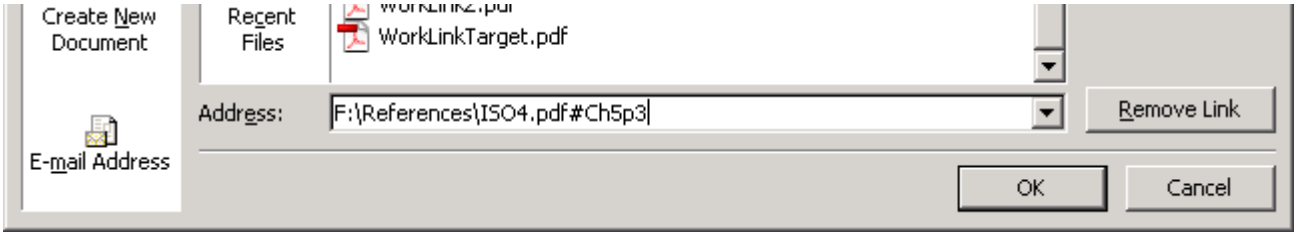


Figure 8-5. Linking an external PDF file with named destination

For the named destination we highly recommend to use the same syntax rules as described in 7.1.4 "How to create a named destination manually" on page 7-88. The "M8.newlink" prefix, however, is not required (but allowed) to be entered. The ND.API plug-in's MS-WORD conversion will automatically add the prefix, if not already contained in the named destination (here *Ch5p3*).

Office 2003

Office 2003 "forgets" the named destination information when converting a document to PDF. Refer to 8.3.5 "Office 2003 does not allow straight ND" on page 8-131 for a solution to this problem.

8.1.4 Link to external MS-WORD document with DOC bookmark

MS-WORD allows to establish a link to a bookmark. In the file dialog (→ Figure 8-2)

1. Select the target file (.DOC) first
2. Press the “Bookmark...” button

The following dialog appears

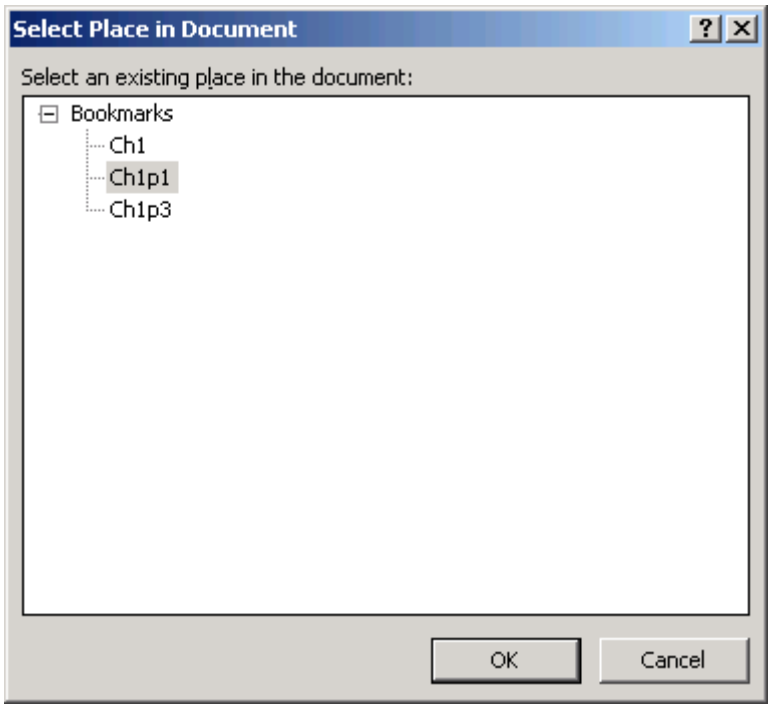


Figure 8-6. Selecting a bookmark in the bookmark dialog

Then MS-WORD adds the “#<nameddest>” notation to the invocation of the file name.

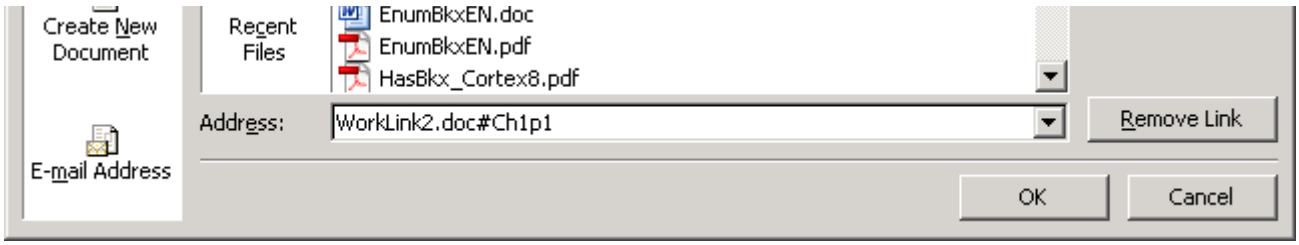


Figure 8-7. MS-Word adds the named destination to the file name

Of course the syntax can also be given right with the file name.

The conversion filter recognizes this situation and will create a proper link to the *named destination* of the target file.

Warning: Using Office 2003 don’t do it this classical way because DOC file links are not converted to PDF targets, which means, you’re doing obsolete work. Instead you should link to stubs. See 8.5.1 “1. Write target with bookmarks” on page 8-134 for further instructions.

8.1.5 Using MS-WORD bookmarks

If you want to set named destinations directly in a WORD document, this is possible by the ***Insert Bookmarks*** function.

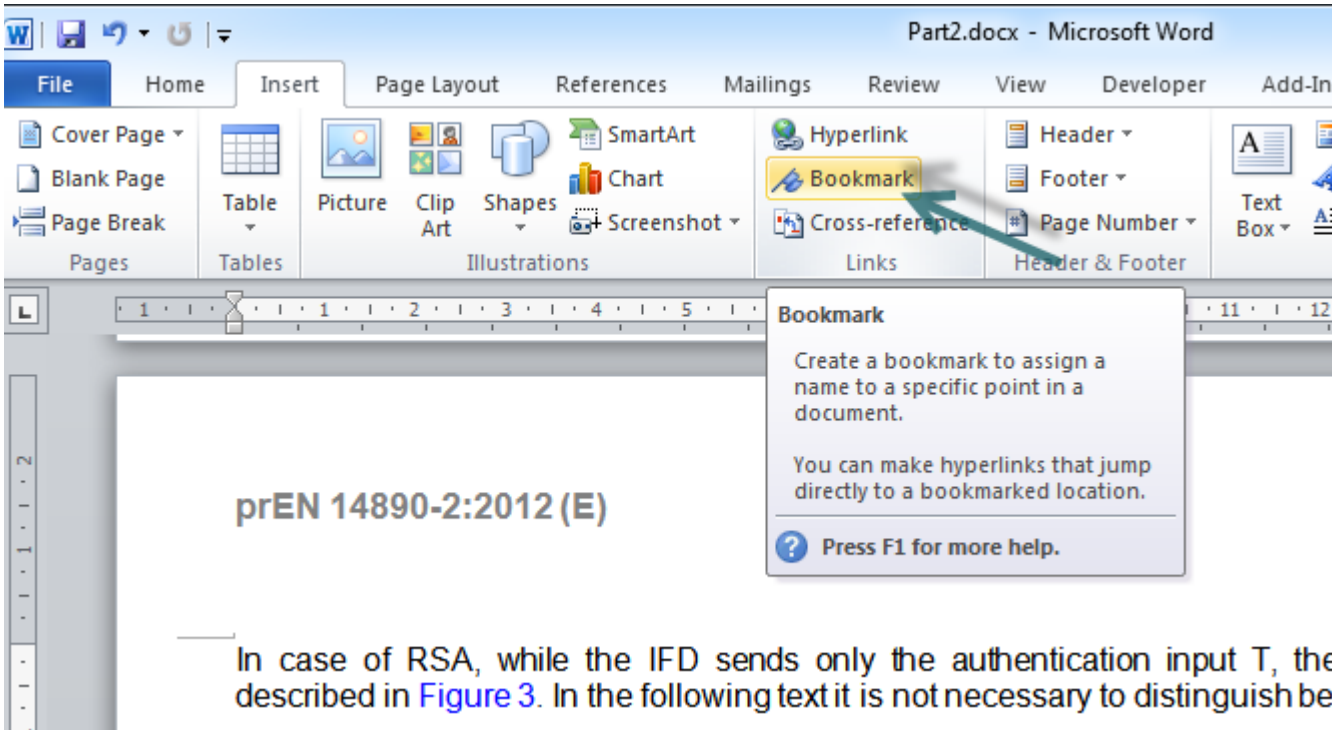


Figure 8-8. Saving as PDF in Office 2010

Such bookmark will automatically becoome a named destination when Printing MS-WORD document as PDF. A dialog will open to ask you for the bookmark name which is identical to the name fo the named desitination.

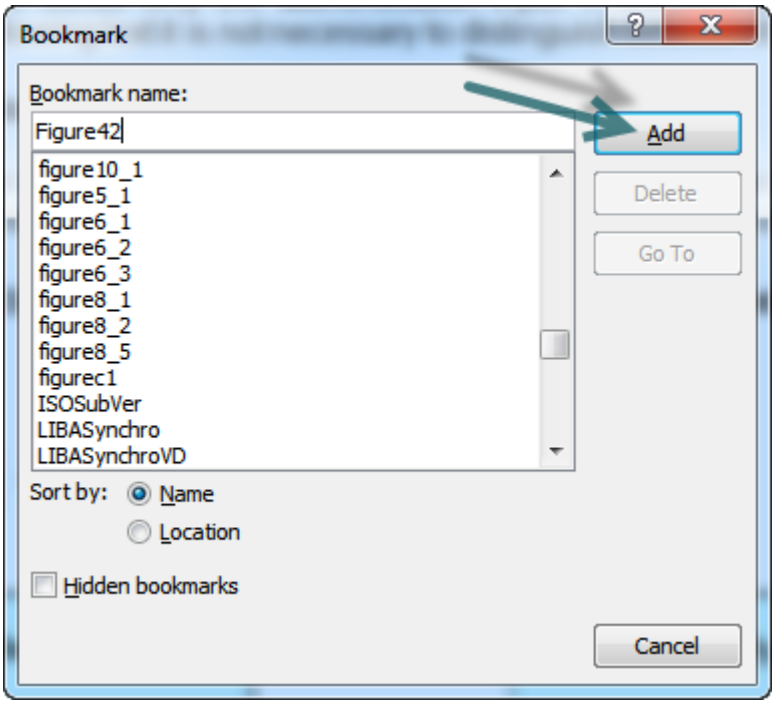


Figure 8-9. Check "Create bookmarks using: "Word bookmarks" in MS-WORD

Choose good names

Do not use spaces, punctuation or special symbols when creating a bookmark. It is highly recommended to use BiCapitalization instead of punctuation symbols. Some symbols are often reserved and can be misinterpeted if special characters are used.

Examples for **good** naming are

Fig07, Tbl03 → uses letter as first char, no spaces, no punctation

Examples for **bad** naming are

Figure 07→ space is not possible

423abc → First letter starts as number, can trigger conflicts

§4.3 → Special char as first character and punctuation

Finally press "Add" in order to set the bookmark.

8.2 Printing MS-WORD document as PDF

Office 2010 has improved the PDF production significantly over Office 2007 and Office 2003. Use the File-Save & Send menu as follows

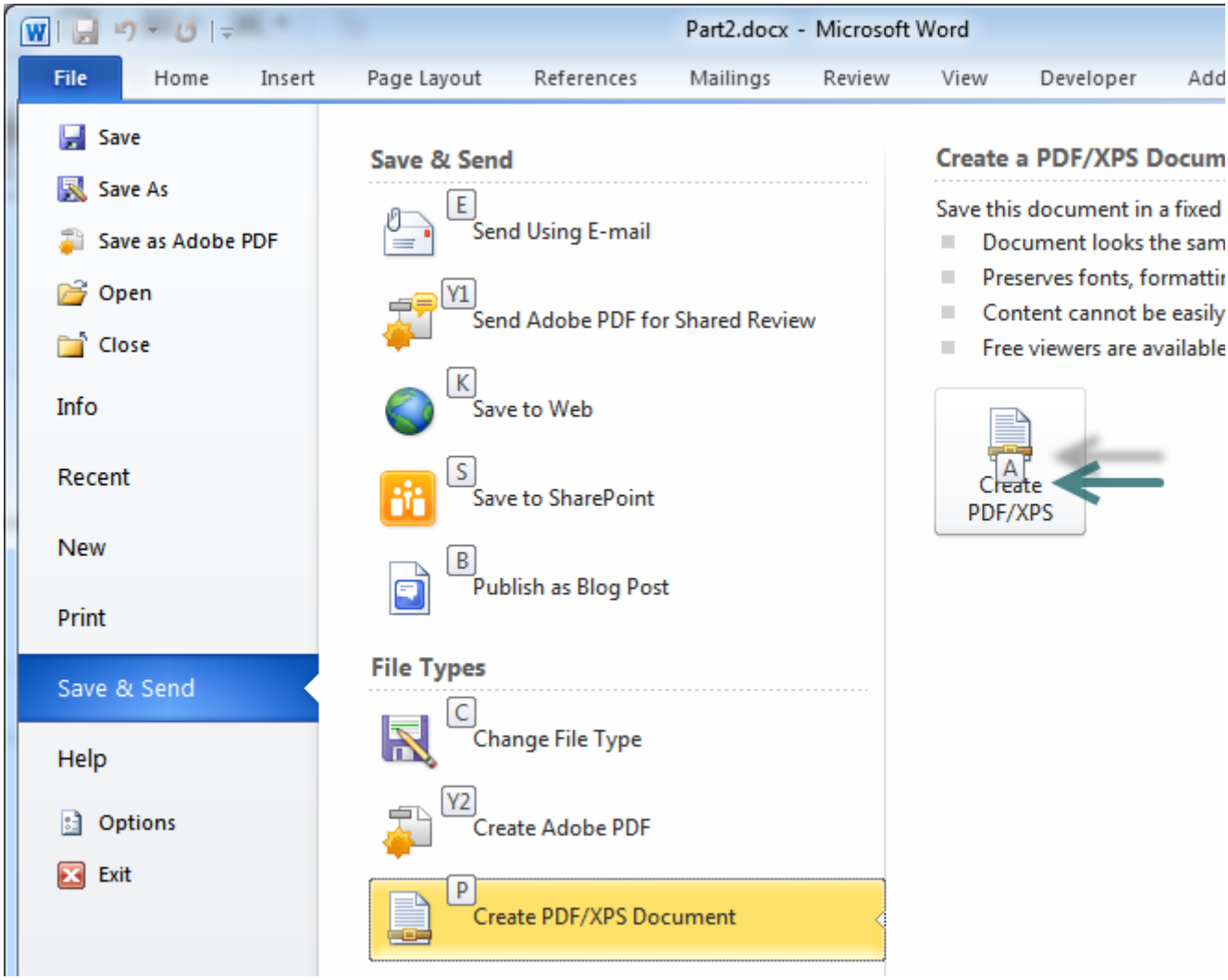


Figure 8-10. Saving as PDF in Office 2010

Saving as PDF is recommended in contrast to the "Print" function because in involves the MS-WORD PDF converter in contrast to the Acrobat Distiller. Nevertheless some artifacts will be generated which can be fixed by the The ND.API plug-in and the Convert URI to stub functions. (see also 8.1.1 “Linking any external document”)

Another print dialog will allow further options - select this button

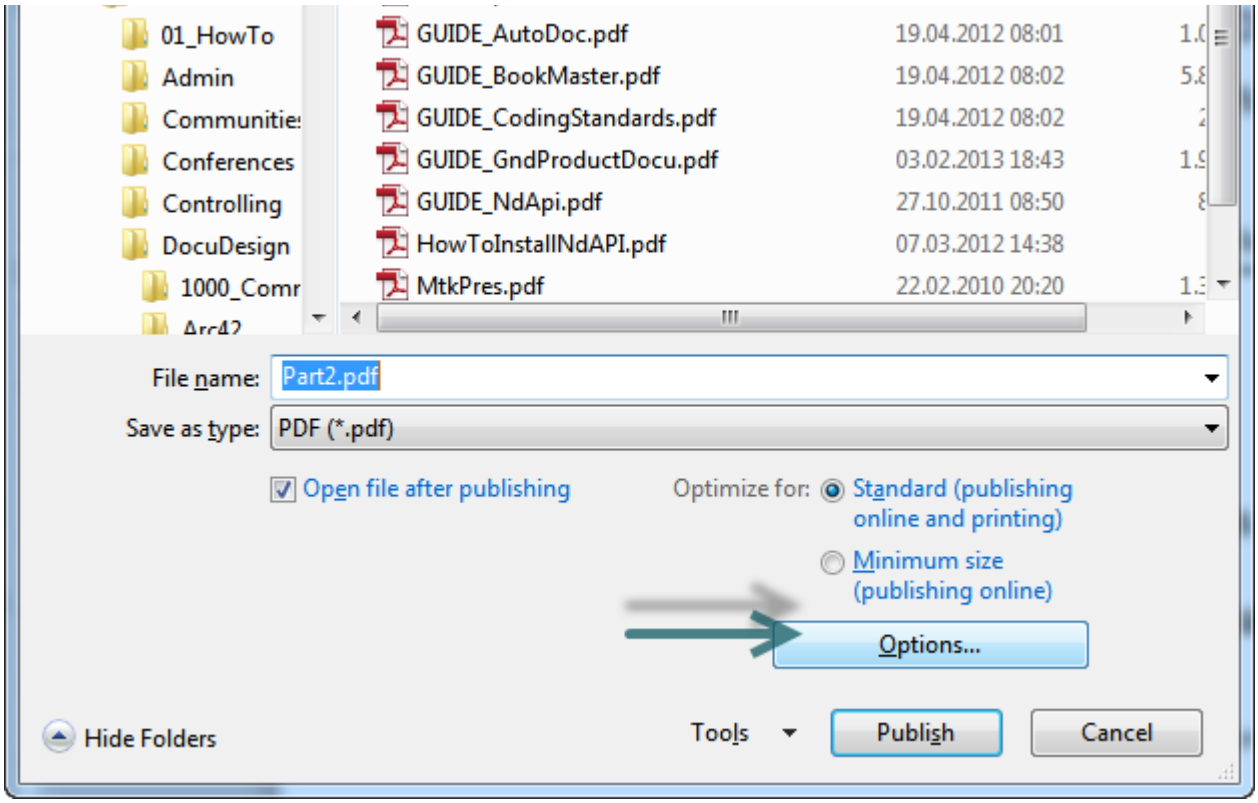


Figure 8-11. Selecting the print options

A dialog box will appear that allows the creation of bookmarks.

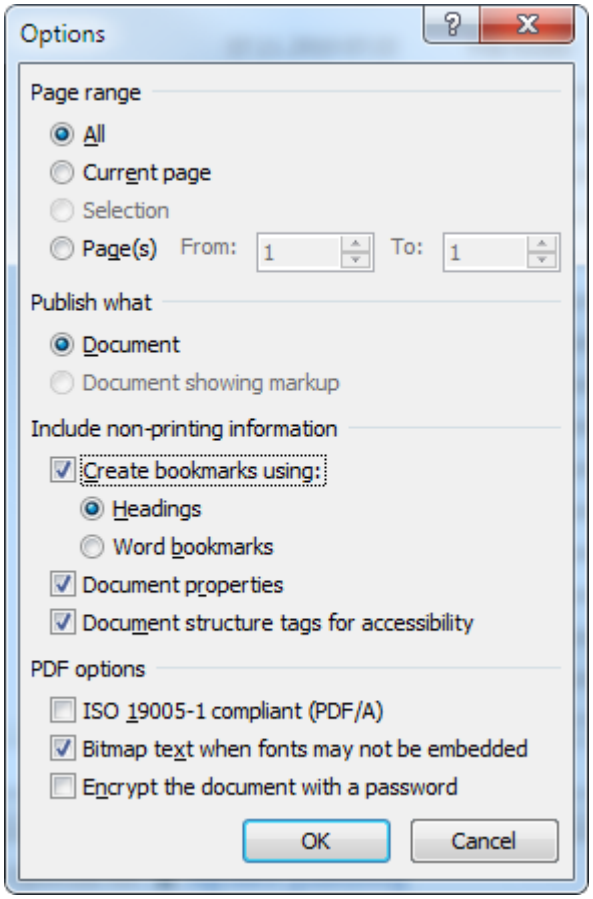


Figure 8-12. Select bookmark creation

Select the creation of bookmarks to get proper bookmarks in the PDF file. Then push the “Publish” button as in Figure 8-11.

8.2.1 Using Bookmarks from MS-WORD

Figure 8-12 shows the selection to create **bookmarks** from **Headings**. This is the default to obtain a correct bookmark list in the PDF file.

It is **not possible at the same time** to obtain named destinations from the *MS-Word bookmarks* in the .DOC(X) file in the same print as bookmarks from headings. If it is desired to get the MS-Word bookmarks too, typically because you intend to link into the printed PDF later from other files) then the same document should be printed again (use another output filename like foo.pdf) with the selection "Create bookmarks using: *Word bookmarks*".

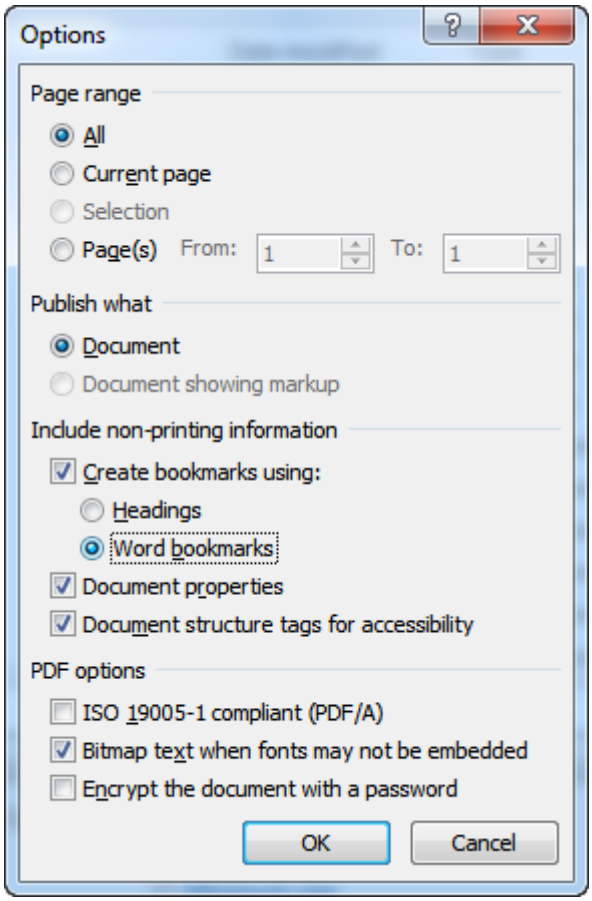


Figure 8-13. Check "Create bookmarks using: "Word bookmarks" in MS-WORD

Printing with Word bookmarks will produce another PDF file that has only the MS-WORD bookmarks in the PDF file. This is certainly not the final result, because now you lack the official heading based bookmarks in the PDF.

However, although the second output file "here: *foo.pdf*" will contain "stupid" PDF-bookmarks, they are important, because these bookmarks allow the creation of named destination – export – and import to the original file. The task can be achieved with a few keystrokes.

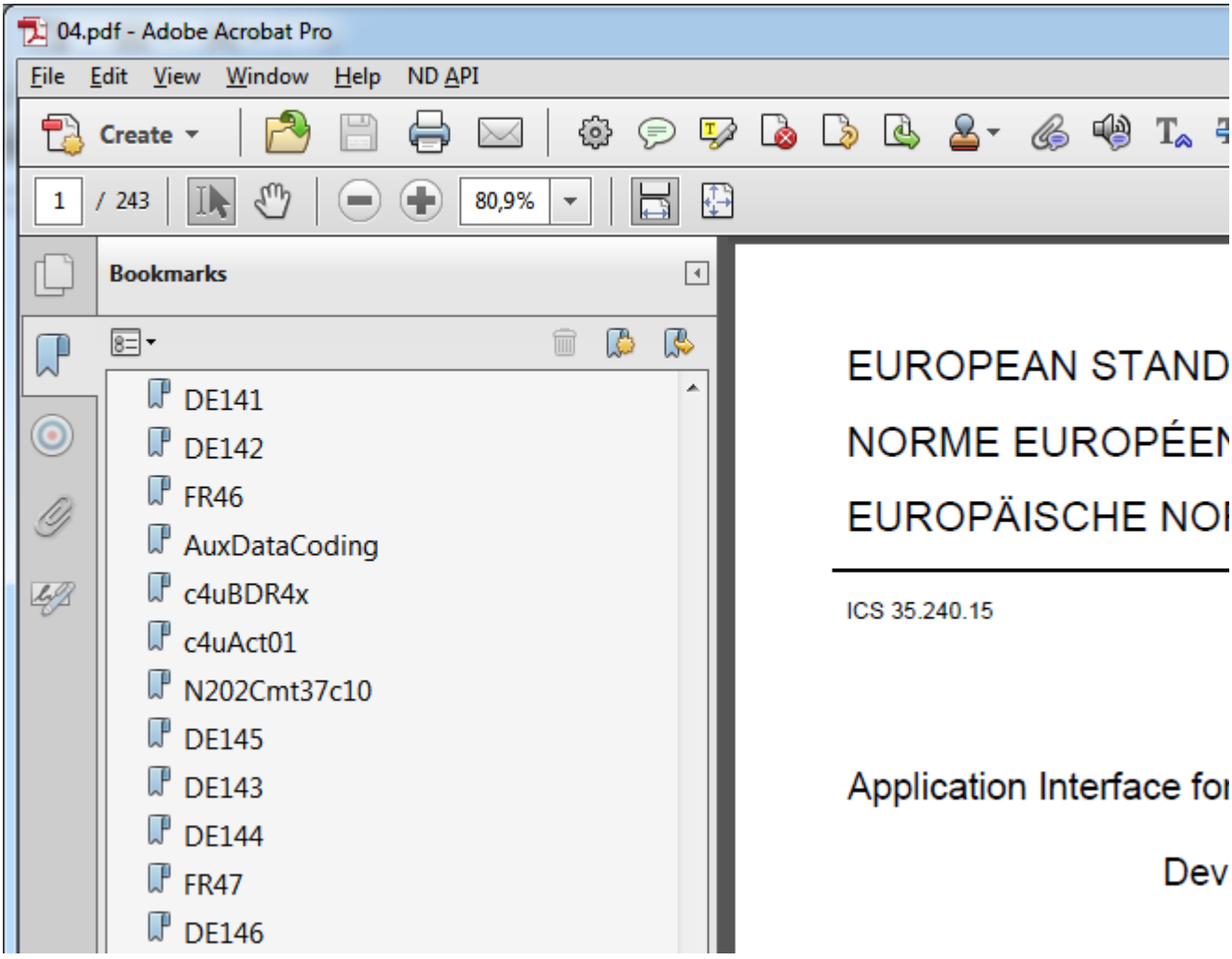


Figure 8-14. Unuseable, but important PDF bookmarks - to be converted to named destinations

Use the function “Generate named destinations from bookmarks” to created named destinations in the output file (e.g. "foo.pdf"). Then **export the created named destinations** using “Export/Import named destinations” in to some "**foo.bkx**".

As a last step, **import those named destinations (foo.bkx)** in the first print of your file which had the option "Create bookmarks using: *Headings*". This is the quickest way to get the named destinations from Word bookmarks.

Ignore the Warning

The import will show an error message indicating that you did not import a BKX file that matches the import PDF, this you can ignore because you actually import into the same print.

8.3 Unwanted artifacts in MS-WORD PDF conversion

Using the “Save As PDF” option in MS-WORD in Office 2003 or Office 2007, 2010, will generate a PDF file. However after the conversion, MS-WORD does not built links correctly. The following artifacts are created in the PDF file

8.3.1 *DOC* file links are not converted to *PDF* targets

MS-WORD Artifact

You can link to external *PDF* files or external *DOC* files but linking to external *DOC* files *does not alter the link* of the source file (linking document) to the PDF representation of the target file (linked document).

That means, if you would click on the link in the SOURCE.PDF (before applying the conversion), it would open MS-WORD to bring up TARGET.**DOC**. Instead it should bring up the TARGET.**PDF**, but MicroSoft tries to bring the reader back to its own products.

Filter Action

The “Convert URI to stub” conversion filter will change the links in SOURCE.PDF such that they point to the PDF representation of the target document.

8.3.2 MS-WORD creates URLs as links

MS-WORD Artifact

If a MS-WORD document was “Saved as PDF” with the MicroSoft plug-in for PDF conversion, then *all links* are generally created as URLs. This, however, is a technical disadvantage because the generated PDF file will always have to open an Internet Browser (typ. Internet Explorer) to view the target file.

Originally MicroSoft might have strongly believe that anything but MS-WORD should be opened in a browser and this made it into the design of links as "always URL".

The consequence of this effect has to be paid by the reader with a very uncomfortable effect.

When an MS-WORD file is converted to PDF, there are some artifacts to be solved post-PDF production.

- MS-WORD does not convert links to DOC(X) into links to PDF when producing PDF
- MS-WORD generates URL links which will open Internet Explorer with a prior prompt to "allow unsafe content". In particular the functions of the Adobe Internet Explorer plug-in is limited compared to the actual Adobe Acrobat (or Reader).

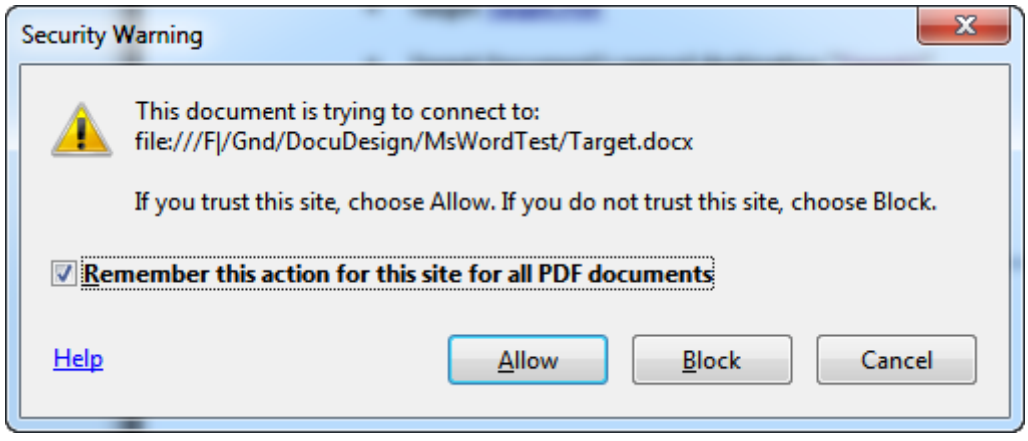


Figure 8-15. Security warning as consequence of an URL based link

On top of this, the Internet Explorer will come up with another response if a DOC(X) was in the original link:

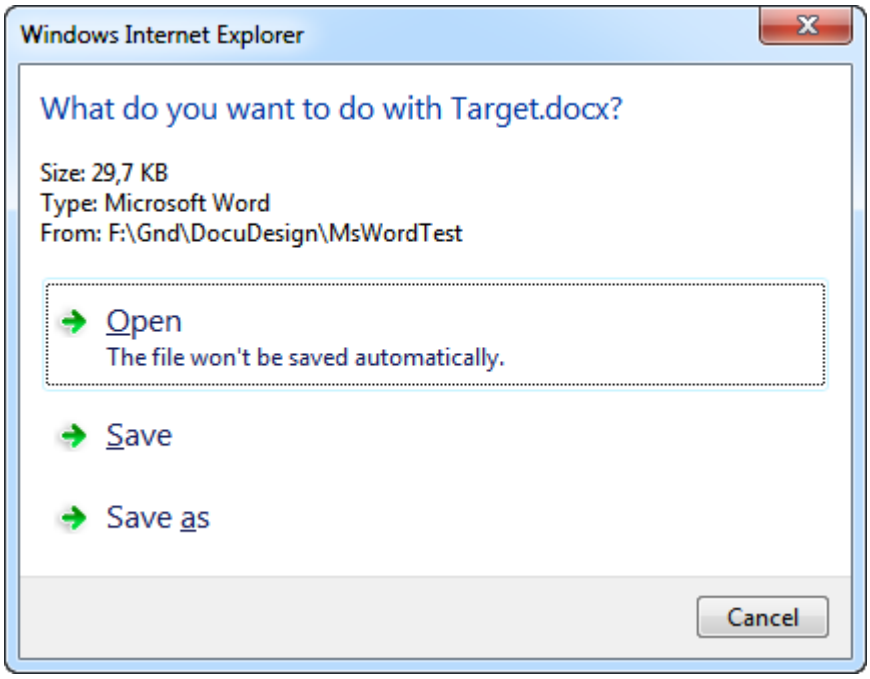


Figure 8-16. Open prompt in Internet Explorer on a DOC(X) link

Open STB file

Opening a .STB file as part of "Stub management" will cause the associated Internet Explorer plug-in to break because a Stub file is an ASCII file associated to a PDF.

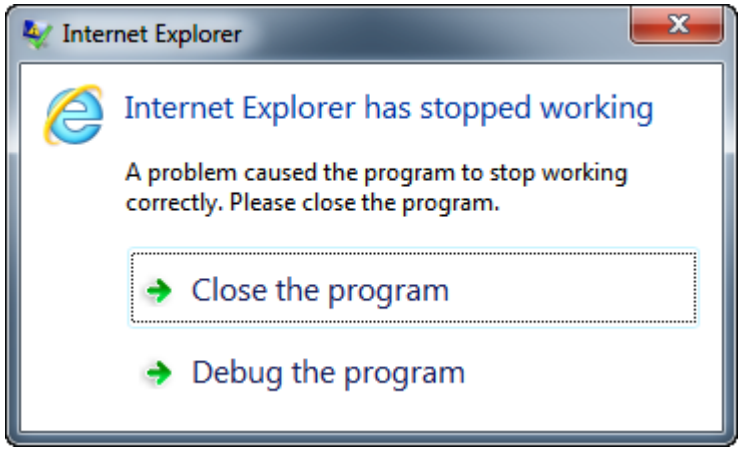


Figure 8-17. Open prompt in Internet Explorer on a DOC(X) link

All these problems are solved by the post-production feature in "The ND.API plug-in" using the functions "Convert URI to stub" or "Convert URI to PDF". Find more information in 12.1.20 "Convert links".

Filter action

The conversion filter converts all the URL links of the SOURCE.PDF into correct Adobe links. The URL type links will be replaced by the correct "GoTo" links and "GoToR" links as specified in the Adobe PDF reference manual. Since they are no longer URLs the Browser won't open anymore but the correct Acrobat Reader instance.

8.3.3 MS-Word produces unstable links

WORD 2003 Artifact

I you ever saved a MS-WORD document in the Acrobat Reader with the "File - SaveAs" function you will recognize, that Acrobat reorganizes the entire PDF file (SOURCE.PDF) to optimize the file size.

In Word 2003 unfortunately *all MS-WORD links get lost by this cleanup procedure*. This is in particular boring for *Table of Content* entries which will *not work anymore* in the "saved-AS" document. The reason for this behavior is unknown, but likely to be found in the MicroSoft design.

Filter action

The conversions "Convert URI to stub" or "Convert URI to PDF" automatically repairs all MS-WORD links such that after a "SaveAs", the links are properly maintained after Acrobat's reorganization.

8.3.4 Office 2003 does not take in bookmarks straight forward

Setting a bookmark in an MS-WORD document might imply that the bookmarks may become named destinations in the PDF file (after converting DOC to PDF). This, however, is another poor MicroSoft design - no only the links to named destinations get lost (→ 8.3.1) but also the associated destinations.

***bookmark** has two-fold meaning*

The meaning of *bookmark* is two-fold. Within the MS-WORD world, it is a particular marker that can be set to a specific position in the word document. WORD documents can link chapters of WORD documents with those *bookmark*-markers.

When MS-WORD converts a file to PDF, then *bookmark* has the meaning of a PDF bookmark which actually shows up in the bookmark tree of a PDF file. At the same time, the internal marker-*bookmarks* get lost.

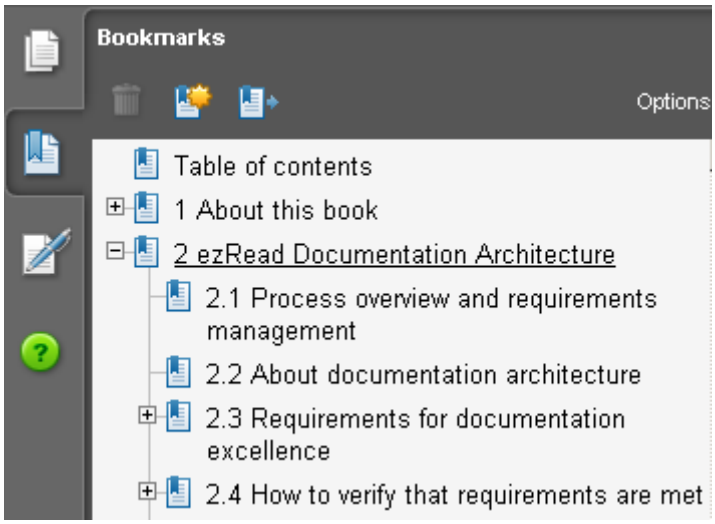


Figure 8-18. These are PDF bookmarks

Taking over WORD-bookmarks to PDF bookmarks

There is a possibility, however, to convert *WORD-bookmarks* into *PDF-bookmarks*. In the *Adobe Conversion Settings* of MS-WORD 2003, you can check the check box "Generate PDF bookmarks from bookmarks". This will generate a special bookmark tree in the PDF file, which does contain the bookmarks.

The additional special bookmark tree is automatically (by Office 2003) named "*Word Bookmarks*".

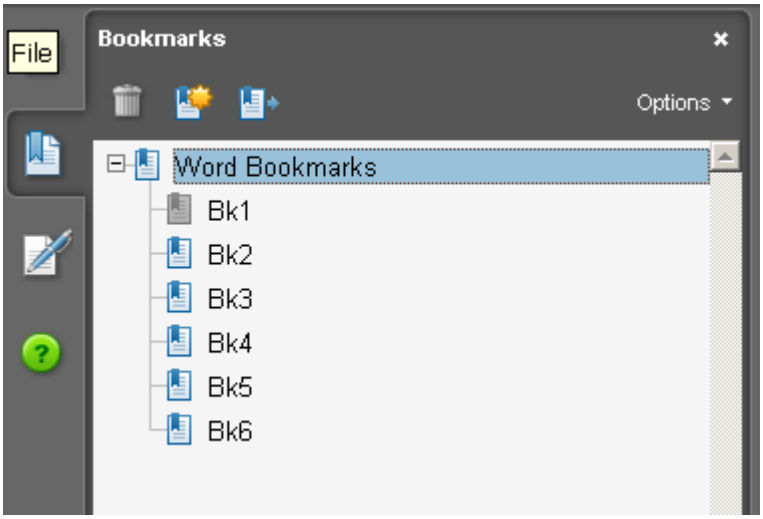


Figure 8-19. MS-WORD bookmarks converted into PDF bookmarks

Until then you still don't have named destinations. That, however is easily achieved through 12.1.6 "Generate named destinations from bookmarks" on page 12-188. There you have the full flexibility. If you want to generate an export file of those imported bookmarks. you may delete all other bookmarks and named destinations first and then you can do the conversion to get the plain imported bookmarks.

8.3.5 Office 2003 does not allow straight ND

MS-WORD Artifact

Office 2003 does not take a link to a named destination into the PDF representation. On Adobe conversion of an MS-WORD document, MS-WORD forgets the named destination and information will not be contained in the "printed" PDF file.

Filter action

Therefore for Office 2003 we recommend Stub management to solve the problem.Instead of referring to a named destination as shown in 8.1.4 "Link to external MS-WORD document with DOC bookmark" on page 8-122, you should refer to a previously generated stub, which is a simple reference to a file instead of the #-namedDest annex.

Converting MS-WORD links as part of The ND.API plug-in's functions, the stub links will be converted in to proper links in the PDF as shown in the chapter below.

8.4 Converting MS-WORD links

The ND.API plug-in offers the functions **Links – Convert URI to Stub/File(MS-Word)**

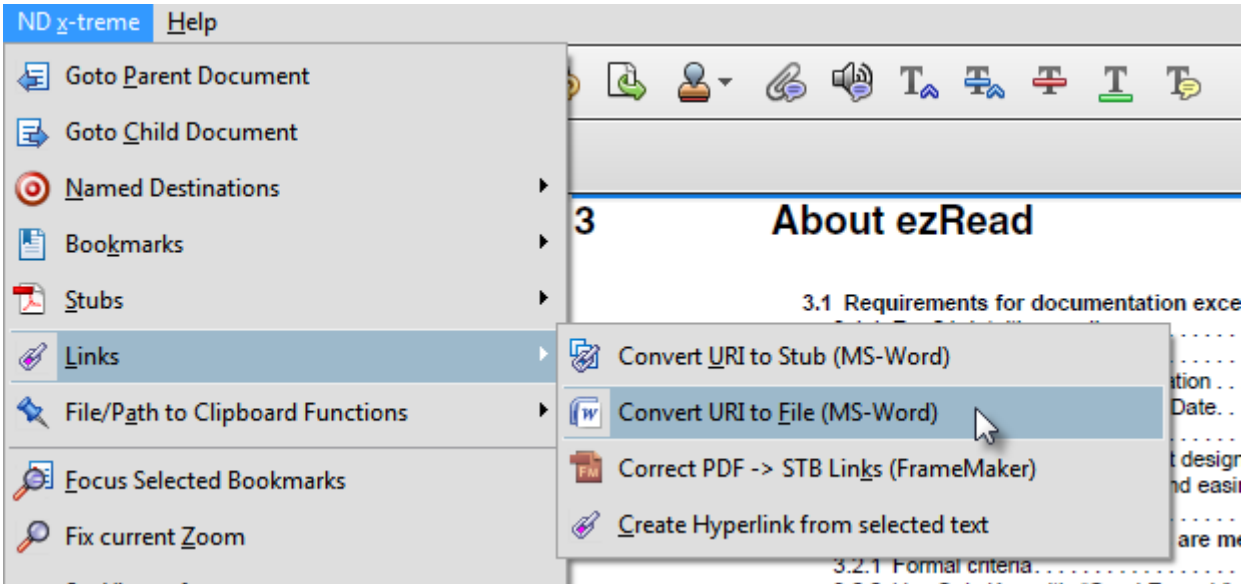


Figure 8-20. Converting the MS-WORD links

Using the conversion

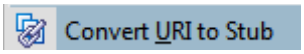
Simply execute the function and save the resulting PDF file. We highly recommend to use the Acrobat "**File-SaveAs**" function to reorganize the document properly.

Converting links to stubs

Of course you can use Stub management also from MSORD documents. This is the easiest way to link to named destinations from MS-WORD documents. The conversion will work according to the following rules

1. If the stub points to an *unsecured* document then a link will be created, that points *directly into the target* - the stub bridge will not be required after the conversion. This will always work as the target document can be equipped with named destinations according to 7.1 "Using Cross References and hyperlinks" on page 7-85.

2. If the stub points to a "Linking FrameMaker to SECURED documents"-type then the created link will again point to the *stub* to allow addressing a particular point in the document.

3. Using 

stubs will always be created regardless from the state of the target document. This is even a recommended method because linking to stubs has some advantages (e.g. target document may change name and location).

Stubs required for SECURED docu

Using the conversion function you don't have to care for this point but you need to know, that for "Linking FrameMaker to SECURED documents" the Stub management has to be maintained also after conversion.

8.5 Use case - from DOC2DOC to PDF2PDF

Because it is a very popular use case, here we give an important example.

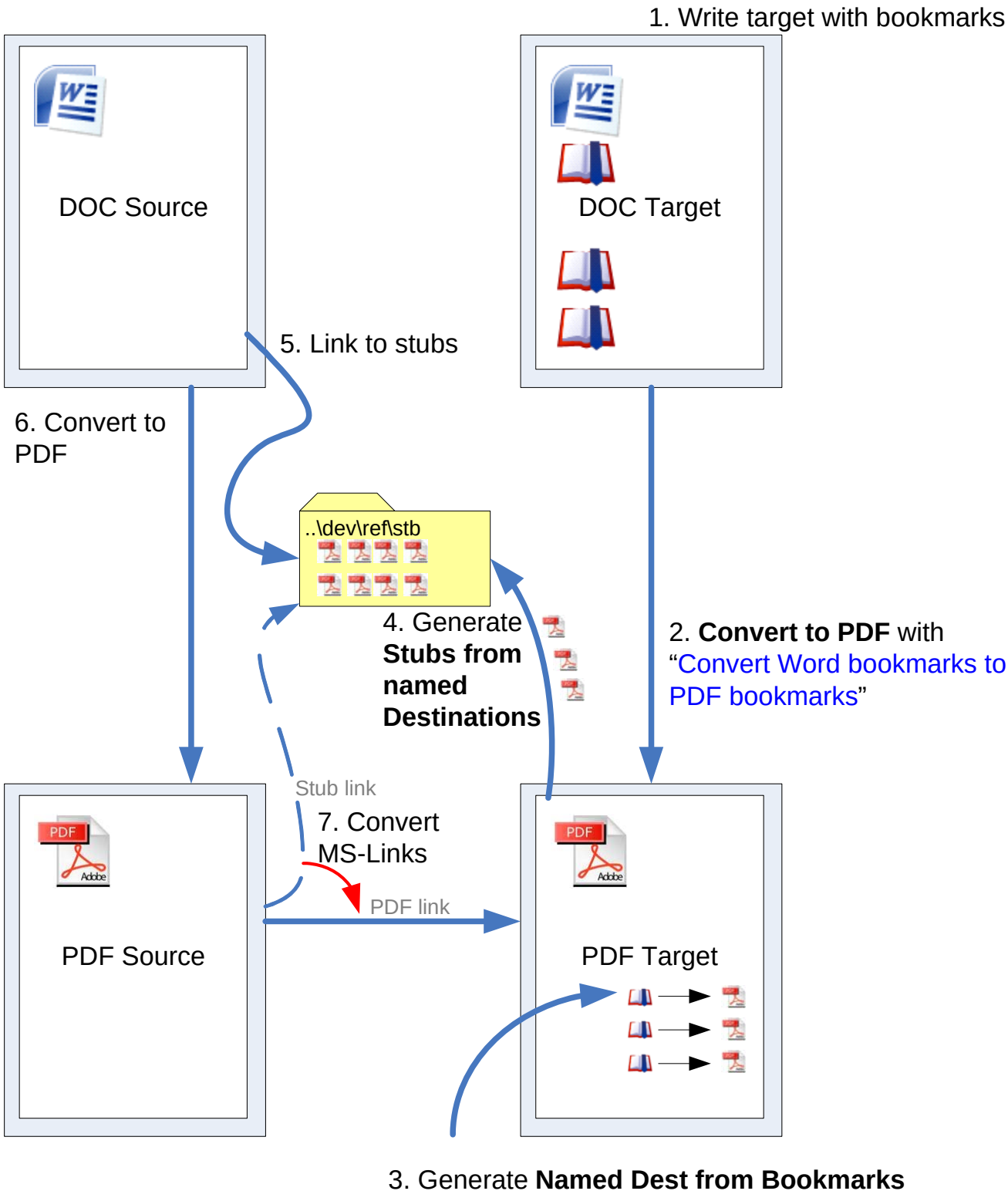


Figure 8-21. Converting the MS-WORD links

Figure 8-21 shows the recommended scenario to get from a DOC-2-DOC link to an appropriate PDF-2-PDF link. Whereas you get all that in one step using Adobe FrameMaker, MS-WORD requires some more attention to get to the same result. Thanks to the "The ND.API plug-in", however, the steps are straight forward and can be applied quickly.

8.5.1 1. Write target with bookmarks

Write the target DOC document. Wherever you need to place a target entry, set a bookmark. This is the usual procedure to link between MS-WORD documents, there is nothing special up to this point.

Don't select when bookmarking

Be careful when setting bookmarks in MS-WORD. I you set a bookmark over a selected range that includes a table i.e. outside/inside of a table is within one selection, on PDF creation MS-WORD generates a bad page number #1 instead of the proper position of the bookmark

The best advice is, not to select anything while defining a bookmark in MS-WORD. Just place the cursor to the desired point in the text - selection of a range doesn't make sense for a bookmark anyway.

Do not use MS-WORD Hyperlinks

Warning: In the *source* document (the one that refers to the target), normally *Hyperlinks* are used to address the bookmarks. The scenario is well explained in 8.1.4 "Link to external MS-WORD document with DOC bookmark" on page 8-122.

However - we don't do it the classical way because DOC file links are not converted to PDF targets, which means, you're doing obsolete work. Instead you should link to stubs, but how to get stubs, we will see in the next steps.

8.5.2 2. Converting to PDF with WORD bookmarks

Our final goal is to make links using Stub management. To obtain stubs from a PDF file, however we can follow the description Generating Stubs from named destinations. And to get named destinations we can use bookmarks and convert using Generate named destinations from bookmarks.

Luckily, we have a possibility to take over DOC bookmarks into PDF bookmarks

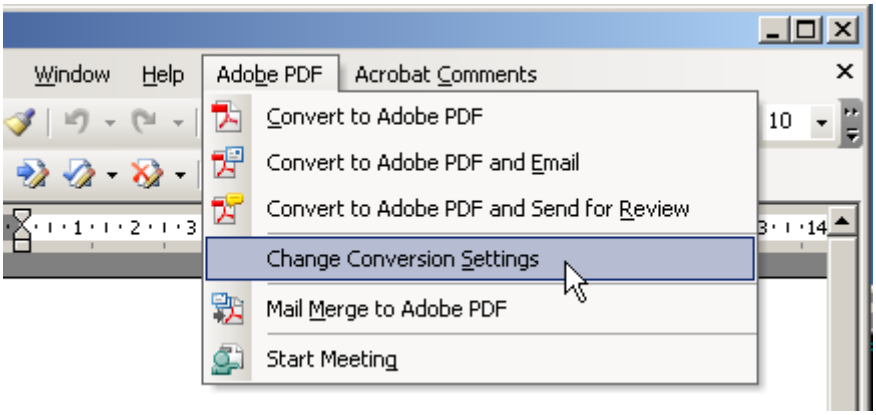


Figure 8-22. Change conversion settings in Word 2003

At first, we have to care that on PDF conversion the MS-WORD bookmarks are taken into PDF bookmarks. Therefore we select “Change Conversion Settings” in MS-WORD 200x.

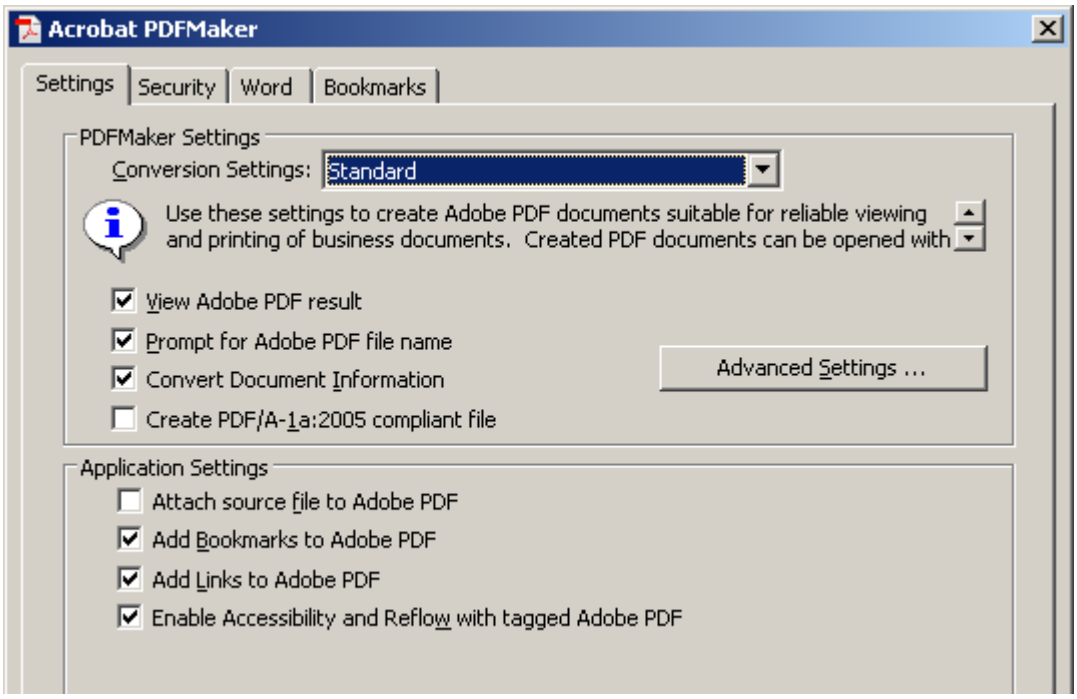


Figure 8-23. General settings

On the general settings check the boxes as shown above. For the “Word” tab, the settings have to be set as follows

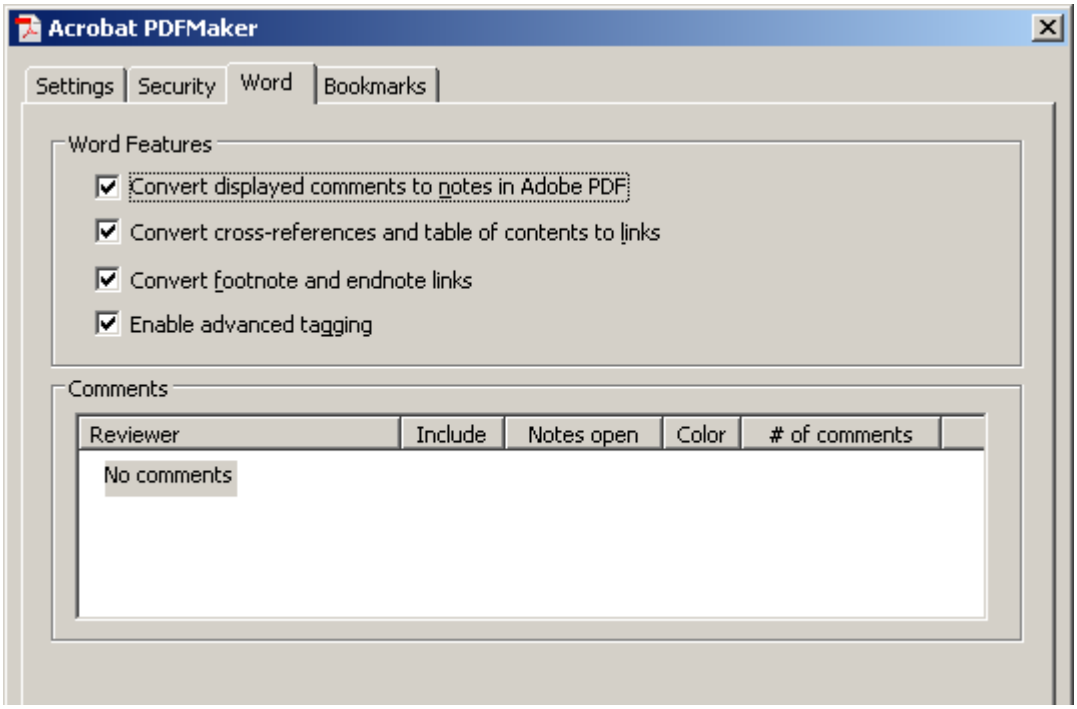


Figure 8-24. Word settings

For the “Bookmarks” tab, the settings have to be set as follows

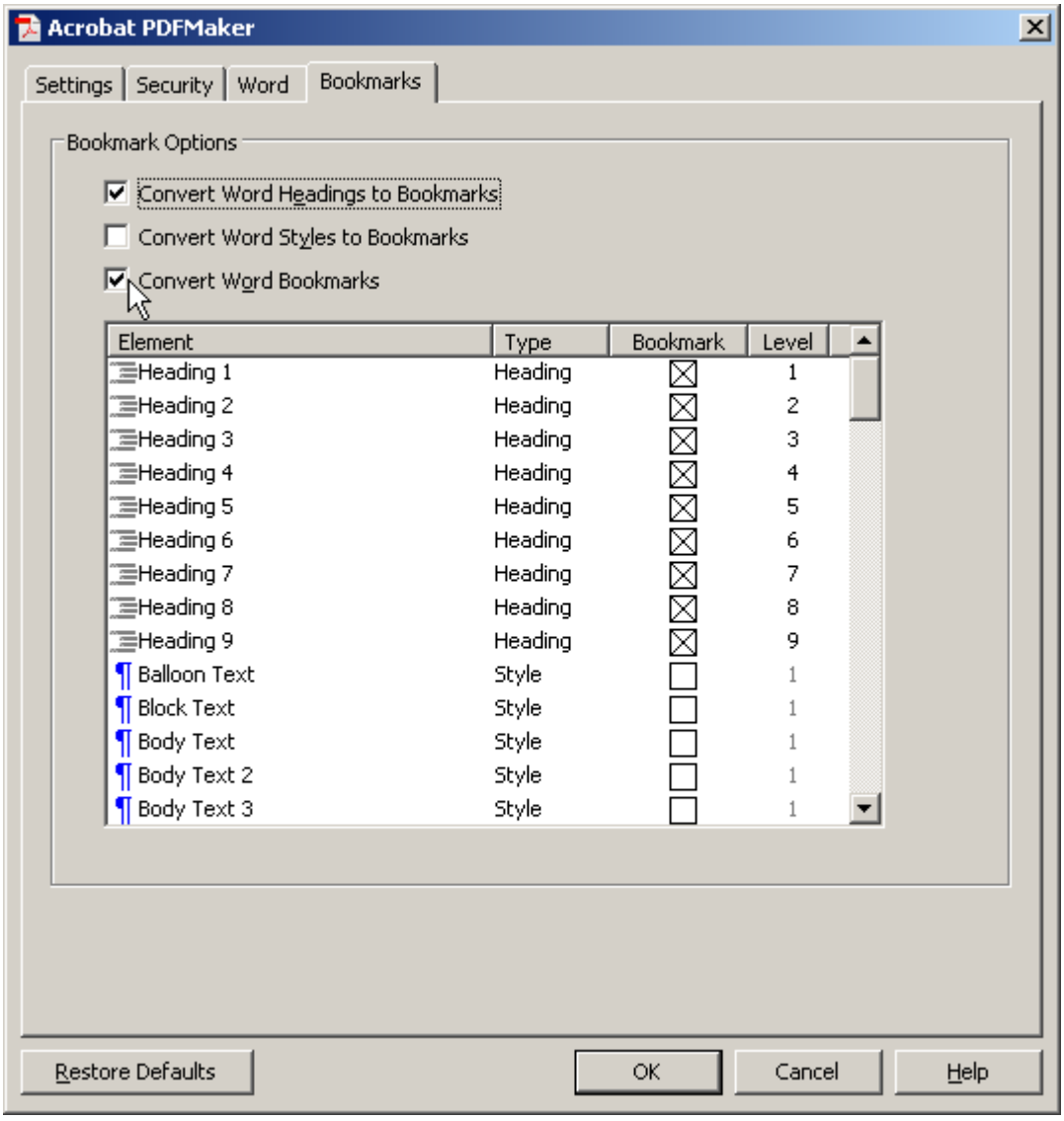


Figure 8-25. Bookmark settings

In particular the check box “Convert Word Bookmarks” is important to be check, otherwise the MS_WORD bookmarks are not taken.

Finally the PDF conversion of the file should be done.

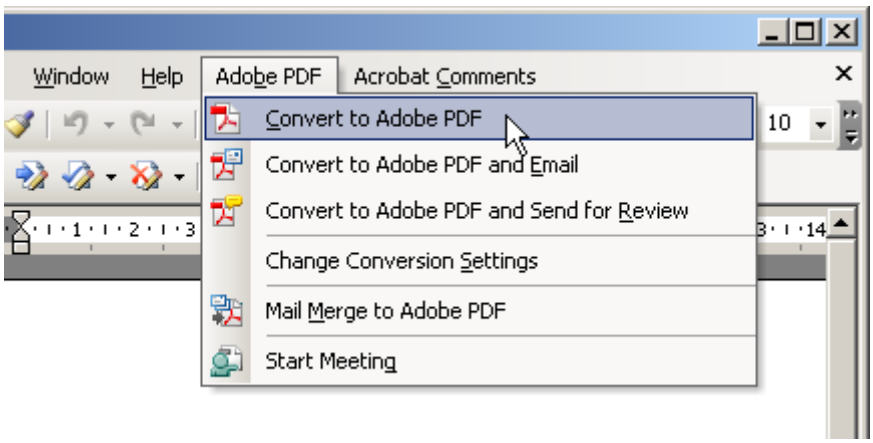


Figure 8-26. Convert DOC to AdobePDF

8.5.3 3. Generate Named Destinations from Bookmarks

The converted target file is now available as PDF. At the end of the PDF Bookmark list it should contain an additional entry "*Word Bookmarks*".

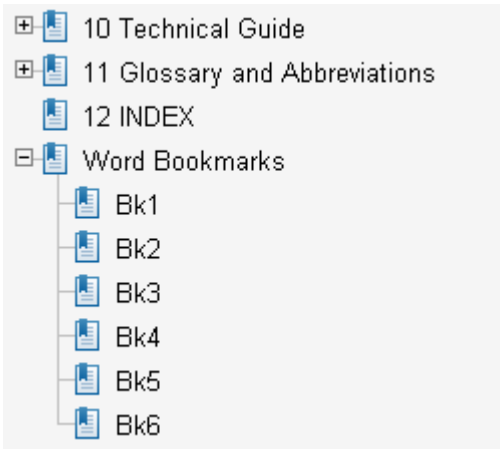


Figure 8-27. Word Bookmarks at the end of the PDF bookmarks should exist

Using Generate named destinations from bookmarks will generate the associated named destinations.

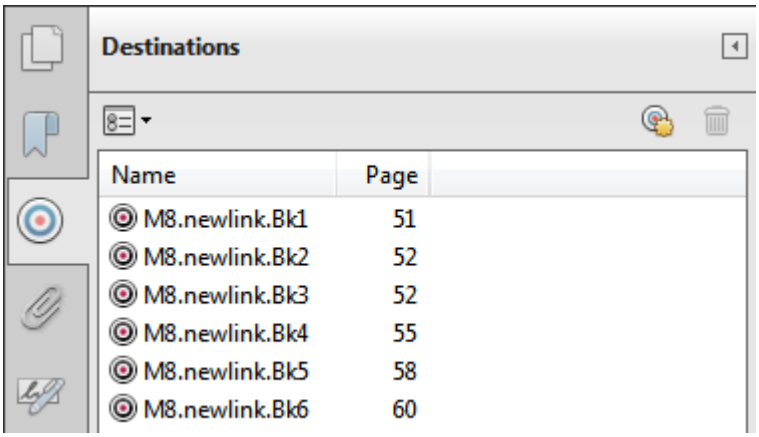


Figure 8-28. Named destinations converted from MS-WORD bookmarks

8.5.4 4. Generate Stubs

Using the function Generating Stubs from named destinations, the first part is done. Stubs are now generated where MS-WORD bookmarks were in the original target document.

Generating stubs is only really necessary when working with Office 2003 because of the MS-WORD artifact that DOC file links are not converted to PDF targets in Office 2003.With Office 2007, it would be possible to address the named destinations right away following 8.1.4 "Link to external MS-WORD document with DOC bookmark" on page 8-122.

Note: However, it is a smart idea to generally work with stubs because, given the stub directory is not moved, it is possible to rename or move the target document elsewhere. Generating the same stubs from the new filename/file position, replaces the link chain whereas the source file does not even notify that change - it still links (working-again) stubs.

8.5.5 5. Link Stubs

Now the source document can use hyperlinks that do not need bookmark links but only file links to stubs as described in 8.1.1 “Linking any external document” on page 8-120. The fact that DOC file links are not converted to PDF targets is no more critical since we do no more try to refer to named destinations from Office 2003.

8.5.6 6. Converting source file to PDF

This is very much the same as 2. Converting to PDF with WORD bookmarks. And why should it be different, in general the source file can be a target file (even of the target file) itself.

8.5.7 7. Convert MS-links

Using Stub management on the “DOC”-level is recommended as explained in step 4.However, when it finally comes to PDF, stubs are not really wanted because they make a lot of files which give little comfort, when a set of documents is to be exported. Whereas the actual export function does have its own stub removal (→ 7.2 “Exporting books and references” on page 7-110) is nevertheless desirable to remove the interim steps via stubs after the PDF2-PDF has built.

Using the function “Convert links” - all links to stubs are replaced by their direct correspondence into the target PDF. This does always work because the target PDF document is not SECURED - which would be the only exception that stubs cannot be replaced.

8.6 Creating bookmarks from comments

Using EXCEL or PowerPoint files, it is not possible to have *bookmark* targets in either of them. But this problem can be solved by comments that follow a particular syntax. Read 12.1.8 “Generate named destinations/bookmarks from comments” on page 12-191 for further information.

9 Storing Reference Documents

9.1 Where to store reference documents 139
9.2 Document naming conventions 139
9.2.1 Using separate convention profiles. 140
9.2.2 Reference Document File Name Format 140
9.2.3 Examples for document naming convention. 142
9.2.4 Document Prefix for typical product documents 142
9.2.5 Typical workflow order 144
9.3 ezRead bibliography naming conventions. 144
9.3.1 Scientific [n] notation. 144
9.3.2 [Abbrev] notation. 144
9.3.1 Scientific [n] notation. 144
9.3.3 [ezRead#9.3.3] notation and syntax 145

9.1 Where to store reference documents

Reference documentation shall be stored in PDF format in one directory, if possible. See 6.4 “Where to put documents” on page 6-70. The reason is, that a receiving party can little be controlled where to place a document package..

> Very often documents are placed in the Windows *My Documents* folder. That however is known to be a bad solution because on operating system crash and related restore, the risk is very high that valuable information disappears. Any project/product related documentation should stay on a partition different from the operating system

To keep complexity low, only two folders are recommended to be used when delivering **ezRead** documentation.

- *Books* folder
- *References* Documents folder

The ND.API plug-in provides a special export function to export a set of linked PDF files into two directories. Find more under 7 “Cross Reference Techniques” on page 7-85.

9.2 Document naming conventions

Document Naming conventions for documents are available as many as projects exist. On nearly every project this topic is discussed again. The following suggestions base on long term experience from many projects and should serve as a base suggestion. Project related conventions may, however, override the recommendations given below.

ezRead requires to store many documents in one reference directory (→ 6.4 “Where to put documents” on page 6-70) because this makes the export significantly more acceptable. Therefore it is important to put smart information into *file names* as they may perhaps serve as an important selection criteria.

9.2.1 Using separate convention profiles

The suggested naming convention uses two compatible profiles. Naming Profile 1 uses a more relaxed syntax, in particular it allows spaces in file names. This responds to the demands of project management and marketing habits where it is common use to use less technical conventions

Naming Profile 2 is more restrictive and intended for technical documents which are subject to processing by script engines. Spaces are not allowed as script engines often might interpret spaces as field separators, e.g. Using auto-generated documents, in particular, doxygen might yield incorrect results when they encounter file names with spaces.

Since marketing and project documentation is not subject to script engine processing, spaces can be afforded in that case.

9.2.1.1 Naming Profile 1

Allows spaces in the Title.

9.2.1.2 Naming Profile 2

Does not allow spaces in the Title.

9.2.2 Reference Document File Name Format

A document file name consists of several parts, which are to be separated by the underscore sign '_'. Optional parts are displayed in brackets. Due to operating system restrictions, no spaces or special characters (e.g. umlauts) are allowed in the file name. The *dash* is allowed, the *underscore* must not be used except as separator to allow tools parse easily through a list of file names.

[Date_] [Prefix_] [BibLink_]<Title> [_Version/Date].<Extension>

9.2.2.1 Date

An *optional* document date shall appear in the document name *only* if the document *refers to a particular event* that was or is to be performed on that date, e.g. meeting minutes, a specific *singular presentation*, agendas etc. In this case, the document date shall be prefixed in the ISO format *yyyymmdd*, e.g. **20081231-PROT-CustomerMeeting.doc**

Note: As hyperlinks and/or cross references are using file names, using a date in the file name will create problems on newer versions of the same document.

Therefore you should avoid *date* and *versions* in any document you want to be changeable without having to rework references later on. Ideas and appropriate recommendations on the versioning problem are given in 6.7 "Version management" on page 6-81.

9.2.2.2 Prefix

The prefix is recommended to group documents of same nature. Chapter 9.2.4 below shows some highly recommended prefixes for typical architecture documents.

On external documents the prefix may reflect the source, rather than its category. If, for instance a set of documents was received from a partner (Example **ARM** company), then it is reasonable to use a prefix that indicates the company's name to allow easier finding of a category of associated documents (e.g. **ARM_Crtx8DtaSht_Cortex8DataSheet.pdf**).

Consequent use of the prefix is also very pratical using the 12.1.17 "Bulk generation of stubs from document title".

9.2.2.3 BibLink

The *BibLink* is a short abbreviation often used in *bibliography references* → [EMV2000] unless they are expressed by numbers → [17]. Using [BibLink] for such entries and having those matched to BibLinks in file names makes it easy to retrieve a file from a known bibliography reference. That of course implies that those BibLink-names shall be *unique* throughout the document tree, a condition that has to be met when using named bibliography entries anyway.

Using ezRead bibliography naming conventions allows an efficient work using the The .STB extension will be associated to the Acrobat Reader (matching the PDF definition) therefore an .STB file will open the Acrobat like a PDF file does.Instant Jump to Stub. Find more about the associated syntax in [ezRead#9.3.3] notation and syntax.

> If you want to avoid the administration of named bibliography entries, of course you can make use of numbered [7] entries. However, unless working with Adobe Framemaker, this can become awkward too, when adding entries into an existing order of a bibliography chapter.

Note: The *BibLink* is also recommended to be used as the identifier of related stubs when using Stub management.

9.2.2.4 Title

For Naming Profile 1 the title should be expressed in BiCapitalization, however, it is also allowed to include spaces and to be written as normal text with free usage of capital format. The dot '.', however is not allowed

PRJ_GrcBid_Project Plan Greece Bid_20090810.DOC

or

Project Plan Greece Bid_1v3.DOC

are valid examples for Naming Profile 1 but would not qualify for Naming Profile 2.

For Naming Profile 2 the title *shall* be expressed in *Camel Case notation* (also known as *BiCapitalization*) (first letter or word upper case, no spaces).

Abbreviations should be considered a single word, i.e. *only the first letter of an abbreviation should be uppercased*. The title should be selected carefully in order to allow for a thematic sorting of documents. It might be useful to achieve a single level hierarchy, e.g.:

SysRS_FndSysReq_AtlasFoundation.doc
AR_ServiceMgmtSpec_AtlasServiceManagement.doc

9.2.2.5 Version/Date

The document version is to be included in the document title *only* if the document is *not versioned using a version control system*. Any official change in the document requires a change in the document version. For details, see the document versioning guideline.

A version is expressed as **#v#v#** (e.g. **4v2v7**) which is different from using a dot '.' (e.g. "4.2.8"). A dot is only used as extension separator and shall not be used anywhere else in the file name.

The version is to be expressed in *major* and *minor* number(s). Only for informal review purposes, the name of the date of the review (8 digits) and/or the short name of the reviewer (login name) might be appended after the version. For formal reviews, the minor number of the documents needs to be changed by each reviewer.

Note: As hyperlinks and/or cross references are using file names, using a version in the file name will create problems on newer versions of the same document.

Therefore you should avoid *date* and *versions* in any document you want to be changeable without having to rework references later on. Ideas and appropriate recommendations on the versioning problem are given in 6.7 “Version management” on page 6-81

9.2.3 Examples for document naming convention

Examples for file names are

Standards

STD_Iso4_Iso7816-4.pdf (no date or version to allow updates)
STD_Pkcs11_PKCS11_2v2.pdf (version:yes to avoid changes in references)

Minutes

20090520_MnOrMapr_MinutesOnOR-Mapper.pdf (No version on minutes)

Architecture

ArcGUI_AppMgmt_ApplicationManagement_2v0.pdf

Document being reviewed in an informal review

AR_CartridgeManagement_0v5_20081017_erhardth.pdf

In the latter case, *the date is not in the first position* because in is considered to be part of the *versioning information* rather than to stand as publication date.

This is an intentional exception from the rules.

9.2.4 Document Prefix for typical product documents

The document prefix helps to sort and recognize the *purpose* of reference documents. For product documentation books the prefixes have already been described in 4.2 “Architecture documents” on page 4-36). Table 9-1 suggests prefix for some standardized reference documentation.

Table 9-1. Document Prefix conventions

Prefixes for Project Management-related documents	
GUIDE	Standardization document or guide document (coding conventions, architecture standards, …
PROT	Meeting minutes (shall have a date prefix)
AGENDA	Meeting agenda (shall have a date prefix)
PR	Presentation
Prefixes for Reference Documents	
StaRS	*Stakeholder Requirements Specification* (formerly URS=User Req Spec, oder “Lastenheft” (German) → 2.3.3 “Stakeholder Requirement Specification (STARS)”
SysRS	*System Requirements Specification* (formerly AURS = “Pflichtenheft” (German) → 2.3.4 “System Requirement Specification (SysRS)”

Table 9-1. Document Prefix conventions

RQ	*Requirement document* In contrast to a technical specification TS, requirement documents are more like a list of tagged statements that comprehensivly cover the customer requirements. Sometimes those documents are also created by requirement management system (e.g. DOORS). Typically, customer requirements should be formulized in a **StaRS** specification
FD	*Feature Document* Combination of SysRS and FS (and optionally TS) for a particular (smaller) feature. Features are particular modules which may be added as new functionality to a system in the context of technical evolution.
TA	*Technical Article* Newspaper or magazine title that supports a particular topic.
TS	*Technical Specification* Technical exploit to specify technical aspects of a feature. Opposite a Technical Specification is created by a customer whereas a Functional Specification is created as a result of architectural design and requirements consideration.
ARC	*Architecture Document*
FS	*Functional Specification* Detailed technical description of the feature implementation
IF	*Interface Description* can be included in FS
IMP	*Implementation documentation* very often auto-generated from source code
STD	*Standards*
GD	*Guideline* Development guidelines, like naming conventions, directory structure, process description. Guidelines are recommendations. If a customer requires an enforcement of organizational measures, they shall be described in a **StaRS** document.

The following table suggests some typical DocLink combinations for architecture documents.

Table 9-2. Examples for typical architecture document prefixes

Prefixes for Architecture and Design Diagrams (Image Files)	
ArcBPM	Business Process Model
ArcTPM	Technical Process Model
ArcREQ	Requirements Diagram
ArcUSE	Use Case Diagram
ArcDOM	Domain Model Diagram
ArcCLA	Class Diagram
ArcERD	Entity Relationship Diagram

Table 9-2. Examples for typical architecture document prefixes

ArcCOM	Component Diagram
ArcDEP	Deployment Diagram
ArcGUI	Graphical User Interface Diagram

9.2.5 Typical workflow order

Although not in the scope of the **ezRead** documentation, the natural work flow for the creation of documents is shown here.

StaRS (customer + G&D) → **SysRS** → **FS (+TS)** → **ARC** → **TS** (+**IF**) → IMP

The marked documents qualify for being implemented in the DOORS environment (→ Figure 2-1).

9.3 ezRead bibliography naming conventions

9.3.1 Scientific [n] notation

Several conventions are out to represent bibliograpy links. The majority of documents uses the classical "[17]" number notation. The advantage is, it is very short and does not much interrupt the reading flow.

The disadvantage is, that a *reader does not have any clue what is meant by the reference* unless s(he) visits the link. If a working hyperlink is attached to such notation, this would help a bit.

Nevertheless there is no notion of the *chapter* and although this can be hidden in the working hyperlink, it at least requires electronic reading to execute the hyperlink.

9.3.2 [Abbrev] notation

Therefore literature has also made use of an abbreviated notation instead of numbering. [EMV2000] is much more intuitive and improves the "ezRead"-feeling of the reader. But chapter notation is missing yet.

9.3.3 [ezRead#9.3.3] notation and syntax

ezRead Documentation System suggests an even richer scheme in order to make best use of abbreviated information.

Addressing chapters

The book abbreviation shall be a *unique* identifier (here "ezRead") followed by a #-hash value and a *chapter number* (here "#8.3"). With an associated working hyperlink this is a very rich method and allows a reader to refer to chapters of a target document even when working with a paper version.

Paper print?

In general we do not recommend printed paper for its environmental impact through paper production and bleeching.

[ ] syntax

The syntax fo build the book prefix mostly relates to intuitive reading. [ISO4#5.2] does not perfectly describe the entire document whereas [ISO/IEC 7816-4, 2003#5.2] interrupts the reading flow although it is much more precise.

Concessions have been made to the fluent reading and the solution which can be memorized better. – which is the shorter solution.

Document identifier

The document identifier (in the example "ISO") shall follow the following rules

Make it same to BibLink : If you name your documents according to 9.2 “Document naming conventions” then it is good advice to keep the BibLink and the Document identifier same. This makes it easier to grep/find documents in directories from knowing already a unique search pattern, the BibLink (as it is the same as the Document identifier).

Do not use spaces: spaces cannot be recognized, you do can hardly know whether you have tow spaces or one, this often leads to non-functionality, so we better avoid any spaces.

Use BiCapitalization: Good practise to help the eye to live without spaces.

Do not use other special signs: Do not use + - & ! : " \ / @ ( ) ... and the entire list of special signs that you can think of. Any tools that later might parse for the [< id >] construct will be easier to build if you avoid those troublemakers of machine processable input.

Do not use version information: if you can avoid it. Your target might change ... do you want to update all your references to the new target? Use version information only if you need to address one particular historical version explicitly.

Avoid underscore: It might be good to use underscore, but be careful ... people do not remember where the underscore was. [Ifx_HwRef79] after some time will be memorized as [IfxHwRef_79] or in other variants. That’s why ’C’ programming gave up on that practise years ago.

Example

So **do** this ...

```
Ifx79HwRef

DocuGd

ISO4

EMV2000

Ifx79HwRef_4v0                  // version 4.0 wanted, _ to improve visibility
```

and do **not** do that

```
infineon HW Reference Manual SLE79          // too long

lfx79 Version 3.1                           // do not use dot nor spaces
                                            // also avoid version
lfx79HwRef(draft)                           // do not use ( )
```

... hundreds of bad examples possible

Using the ezRead bibliography naming conventions allows an efficient work with the The .STB extension will be associated to the Acrobat Reader (matching the PDF definition) therefore an .STB file will open the Acrobat like a PDF file does. methods described in 7.1.12 “Instant Jump to Stub” on page 7-109.

10 Verifying documentation consistency

10.1 Contents and Scope of this Document . . . 147
10.2 Priorities . . . 147
10.3 Acceptance Criteria for Architecture Documentation . . . 149
10.3.1 Formal Criteria . . . 149
10.3.2 Criteria for understandability of documentation . . . 150
10.3.3 Criteria for architecture goals and constraints . . . 151
10.3.4 Criteria for Context View . . . 152
10.3.5 Criteria for building blocks view . . . 152
10.3.6 Criteria for Runtime View . . . 154
10.3.7 Criteria for deployment view . . . 155
10.3.8 Criteria for Design Decisions . . . 156
10.4 Acceptance criteria for architecture . . . 157
10.4.1 Related to product quality metrics . . . 157
10.5 Acceptance criteria for development . . . 158
10.5.1 Source code artifacts . . . 158
10.5.2 General coding conventions . . . 158
10.5.3 Related to JAVA coding conventions and styleguides . . . 159
10.5.4 Related to Cocoon & XSLT Coding conventions and styleguides . . . 161
10.5.5 Criteria related to Version Control . . . 161

10.1 Contents and Scope of this Document

This chapter describes *acceptance criteria* for *software* and *system architecture*, implementation and their documentation. These criteria shall ensure that evaluators or auditors can trace wether both business and architectural (quality) requirements have been met.

These criteria shall be met by all releases of an architecture and the corresponding documentation, to create a common understanding between all stakeholders involved in system and software development.

A brief outline of the contents:

- Chapter 10.3contains acceptance criteria for architecture documentation. It covers all structural elements of the arc42 documentation template, e.g. the different views, design decisions and general criteria for modelling and documentation.
- Chapter 10.4names acceptance criteria for the architectural structures.
- Chapter 10.5 covers criteria for development, e.g. coding conventions, handling of sourcecode artifacts and usage of version control.

Criteria are described by both an *explanation* (describing the criterion) and a *verification* (how to check). In practice, most criteria have to be checked or verified *manually*. Where appropriate, references to related criteria are included.

10.2 Priorities

Criteria are marked with priorities **A**, **B** or **C**, which carry the following semantics:

A. *Most urgent:* Shall be fulfilled/completed at any time.

B. *Important*, but not very urgent: Shall be fulfilled/completed within at least 6th month from evaluation time.

C. *Long-term goal*, shall be fulfilled/completed in the long run.

10.3 Acceptance Criteria for Architecture Documentation

This chapter summarizes acceptance criteria for *architecture documentation*.

10.3.1 Formal Criteria

10.3.1.1 Criterion 1-1: (Prio A) Use agreed-upon document formats and tools

The architectural model shall be maintained and delivered as a set of UML diagrams (Enterprise Architect)

The documentation shall be written in FrameMaker and be organized according to book according to 4.1 “Book organization” on page 4-35.

Images and other source material as well as the UML diagrams shall be stored according to 9.1 “Where to store reference documents” on page 9-139.

Verification: Inspect that corresponding files are included in release.

10.3.1.2 Criterion 1-2: (Prio A) No structural modification of documentation

Explanation: The structure (sections, subsections) of the architecture documentation shall not be modified without prior written consent.

Verification: Inspect the headings level 1-3 in current version remain unchanged compared to reference version.

10.3.1.3 Criterion 1-3: (Prio A) Architecture documentation per release

Explanation: Every release shall contain a corresponding architecture and design documentation. No exception.

Verification: On a release, check that documents (PDF version of the books) have been delivered. This action is part of the project management.

10.3.1.4 Criterion 1-4: (Prio A) ArchDoc - Release Notes & Change History

Explanation: Changes shall be recorded in the requirements management system.

When publishing a modified version or release, a (detailed) change history or release-note shall be included as supporting document. This change history shall briefly state the WHAT, WHY and WHERE of (major) changes.

Note: This claim is relevant for project management. Product documentation shall be synchronized (→ 3.1.4 “Req04: Accurate and Up-To-Date” on page 3-23) but the change history shall be kept outside product documentation.

Verification: Reference to appropriate project documentation - Manual inspection

- A release-notes document is included in the release.
- For every new or modified requirement of this release, one entry exists in the release note.
- For every bug/issue fixed, one entry exists in the release note.

10.3.1.5 Criterion 1-5: (Prio A) Keep Requirements traceable in architecture and code

Explanation: *Solutions* (implementation) of new or modified requirements shall be easily traceable. This can be done by referencing such requirements in the Design Decisions section of the architecture documentation.

Note: A list-of-changes is relevant for project management. Product documentation shall be synchronized (→ 3.1.4 “Req04: Accurate and Up-To-Date” on page 3-23) but the change history shall be kept outside product documentation.

Verification: Inspection of *project* documentation that a release covers changes in *source code* and *product* documentation.

Related Criteria: Criterion 1-6: (Prio A) Maintain Index of Important Topics

10.3.2 Criteria for understandability of documentation

These criteria focus mainly on the product documentation.

10.3.2.1 Criterion 1-6: (Prio A) Maintain Index of Important Topics

Explanation: An index of keywords improves *findability* and *navigability* within technical documents (→ N.9 “INDEX” on page 5-61).

Maintain an index of all major and important topics within the architecture documentation. The following topics shall be included in the index:

- The names of all building blocks, relationships, associations, nodes and channels
- The names of major code artifacts.
- All *architectural aspects*, keywords and patterns.
- All major *nonfunctional requirements* (their definition and decisions concerning their solution)

Verification:

- Inspection for completeness of the index This is typically chapter related and shall always be maintained by adding the appropriate markers when a chapter is finished.
- Record missing index-entries as (low-prio) issues/bugs for next release.

Related Criteria: none

10.3.2.2 Criterion 1-7: (Prio A) Include version information in diagrams

Explanation: Every diagram shall contain a version information or *last modification date*, usually by stating the modification date. (→ 6.2.1 “Include time stamps to architectural images” on page 6-66).

See Figure 10-1 for an example.

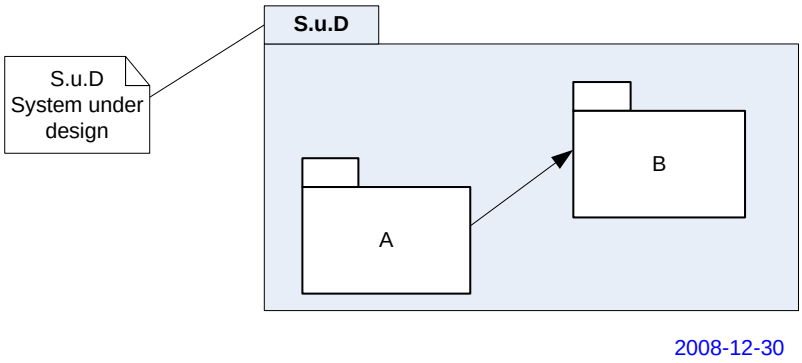


Figure 10-1. Sample UML Diagram with Version Info and Legend

Verification: Inspection.

Related criteria: Versioning of all artifacts, commit-comments

10.3.2.3 Criterion 1-8: (Prio A) Legend in all diagrams for non-Std UML constructs

Explanation: Diagrams which use non-standard UML constructs (e.g. colored symbols or stereotypes) shall include a *legend* describing these non-standard elements. Conventions can be made global by declaring or describing them at appropriate locations within the documentation. In building-block diagrams this shall be done by including a „Legend“ package. In runtime views, this is mostly impossible due to restrictions in UML.

Verification: Inspection.

Related Criteria: none

10.3.2.4 Criterion 1-9: (Prio A) Textual Description of Non-Standard UML constructs

Explanation: Sequence or activity diagrams (or other runtime diagrams) which do not allow a legend within the diagram shall have a textual addendum describing non-standard UML usage (color etc.).

Verification: Inspection.

Related Criteria: <t.b.d>

10.3.3 Criteria for architecture goals and constraints

10.3.3.1 Criterion 1-10: (Prio A) Describe current goals and constraints

Explanation: Architectural changes between different releases shall be traceable to goals, requirements or constraints. Therefore the sections on goals and constraints within the architecture documentation (which are usually extracted from the appropriate versions of requirements documentation) have to be kept up-to-date in all releases.

→ N.1 “Introduction and Goals” on page 5-54.

Verification: Inspection

10.3.3.2 Criterion 1-11: (Prio A) Describe current stakeholders

Explanation: The list of stakeholders shall be kept current. This list comprises of stakeholder-roles only, no names of persons shall be mentioned (unless explicitely required).

→ N.1.2 “Stakeholder” on page 5-54.

Verification: Inspection.

Related Criteria: Criterion 1-10: (Prio A) Describe current goals and constraints

10.3.4 Criteria for Context View

10.3.4.1 Criterion 1-12: (Prio A) Document complete system context

Explanation: The context views shall be exhaustive and complete. All neighbour systems shall be depicted and (briefly) described here.

→ N.3 “Context View” on page 5-55.

Verification: Inspection. Ensure that every IT-system directly related to the architectural topic is named and described in the context view.

Related Criteria: Criterion 1-13: (Prio A) Document All External Channels

10.3.4.2 Criterion 1-13: (Prio A) Document All External Channels

Explanation: All channels leading in and out of the system shall be descibed. No exception allowed.

→ N.4.1.1 “Building Block 1 (Blackbox Description)” on page 5-58.

Author’s Note: ToDo: Elaborate on required level of detail.

Verification: Inspection.

Related Criteria: Criterion 1-12: (Prio A) Document complete system context

10.3.5 Criteria for building blocks view

10.3.5.1 Criterion 1-14: (Prio A-B) Describe every building block

Explanation: Every building block plus every association between building blocks shall be described, at least to package level. No exception.

Prio is A for all batch jobs, B for web applications.

→ N.4 “Building Block View” on page 5-56.

Verification: Inspection. Cross-check with deployment view: Every deployment artifact (jar, ear, war or similar) shall be traceable to a building block.

Related Criteria:

10.3.5.2 Criterion 1-15: (Prio A-B) Use diagrams to describe whitebox-views

Explanation: Whitebox views shall be described using either UML packet or component diagrams. Color might be used to express characteristics. For Cocoon applications:The structure of the sitemaps shall be explained (in terms of components, resources) etc. Prio is A for all batch jobs, B for web application.

→ N.4 “Building Block View” on page 5-56.

Verification: Inspection

Related Criteria:

10.3.5.3 Criterion 1-16: (Prio A) Describe blackbox purpose, interfaces and variability

Explanation: Blackbox descriptions shall at least consist of purpose and responsibility, interfaces, and variability.

→ N.4 “Building Block View” on page 5-56.

Verification: Inspect.

Check that every building block contained in the architecture documentation includes the following:

- a (brief) purpose.
- an *interface description* in useful level of detail.
- an *explanation* of its variability (or „none“).

Related Criteria: none

10.3.5.4 Criterion 1-17: (Prio A) Describe code artifacts mapping of building blocks

Explanation: On detailed levels of building-block descriptions, the major artifacts needed to implement the building block shall be referenced in *section location/important files*.

The latter section should contain the *main entry point* (e.g. main class or classes, sitemap etc.) plus brief info on dependent artifacts (like build files). Additional information is optional.

→ N.4 "Building Block View" on page 5-56.

Note: Refrain from giving too many details here. Usually developers use their development environment to determine the current and complete set of required artifacts, they do not consult written documentation for that purpose. The actual documentation shall be found in the 4.2.6 "IMP - Implementation documentation" on page 4-40 which is generated automatically from source code.

Sample: EdifactImporter (from Building Block View, Detail Level 2). The Edifact Import Job stores documents that have been supplied by <X> in the <SUD> database. **Location & Files:**
Source: Main class: org.arc42.rules.RuleProcessor.java Important dependencies:

- LocatorMessage: Holder structure for an locator message. Populated by Spring.
- DBConnection: Transfers an instance of LocatorMessage to the database.

Buildfile & Dependencies: Build with (ant-based) build.xml.

Verification: Inspection. Building blocks on detailed levels (≥ 3) shall have their „Location & Files" section filled in.

Related Criteria: none

10.3.6 Criteria for Runtime View

10.3.6.1 Criterion 1-18: (Prio B) Document runtime scenarios

Document use cases, external protocols and error conditions as runtime scenarios

Explanation: Runtime views should be documented for the following scenarios:

- All major use-cases.
- All external interfaces with their protocols (import/export to external systems).
- Error conditions.

Means for documentation shall be diagrams plus (textual) descriptions, plus references to the appropriate building-blocks.

→ N.5 "Runtime View" on page 5-60.

Verification: Inspection

10.3.6.2 Criterion 1-19: (Prio B) Document Typical Change-Scenarios

Explanation: Runtime views should be documented for typical and expected change-scenarios: How can typical changes (between releases) be incorporated into the system (e.g. changes required to include a new use-cases).

→ N.5 “Runtime View” on page 5-60.

Verification: Inspection

Related Criteria: Criterion 1-18: (Prio B) Document runtime scenarios

10.3.7 Criteria for deployment view

10.3.7.1 Criterion 1-20: (Prio A) Describe deployment artifacts

Explanation: Every deployment artifact (that is every file which is used at runtime) is described. No exception.

→ N.6 “Deployment View” on page 5-60.

Verification: Inspection.

Related Criteria: Criterion 1-21: Associate deployment artifacts with building blocks

10.3.7.2 Criterion 1-21: Associate deployment artifacts with building blocks

Explanation: Every deployment artifact shall have at least one associated building block from the building block view. This association is not necessarily a 1:1 mapping, as

- a single building block might yield several deployment artifacts (e.g. one jar-file plus one configuration property file)
- a single deployment artifact might have several building associated building blocks (e.g. a whole package of java-files compiled into a single jar-file).

Deployment artifacts can be binary (jar, war, ear, zip) or textual files (e.g. xml, xsl, xmap, txt, config).

→ N.6 “Deployment View” on page 5-60.

Verification: Two-step verification:

1. Ensure that released deployment artifacts match the documented deployment artifacts.
2. Inspect if every deployment artifact has at least one associated building block.

A good source to identify deployment artifacts are build-files (ant, maven, make).

Related Criteria: none

10.3.7.3 Criterion 1-22: (Prio A) Include configuration files in deployment artifacts

Explanation: Especially for Cocoon and similar frameworks: The major configuration files (especially sitemaps) used as deployment artifacts shall be named.

Exception: It is not necessary to list all stylesheets in the architecture documentation).

Verification: Inspection.

Related Criteria: Criterion 1-21: Associate deployment artifacts with building blocks

10.3.8 Criteria for Design Decisions

10.3.8.1 Criterion 1-23: (Prio B) Document major design decisions

Explanation: Major design and architecture decisions shall be documented with the following details:

- Reasons for this decision.
- Alternatives considered but rejected (plus reason for rejection)

→ N.7 “Design Decisions” on page 5-60.

To decide which decisions fall into this category, consider the following examples:

- *Introduction* of new building blocks on *detail-level 2 or 1*.
- *Structural modification* on building-block detail level 2 or above.
- Decisions concerning system operations or administration, e.g. new deployment requirements, modified error/exception-handling, usage of new libraries, 3rd-party-components etc.

Verification: Inspection.

Related Criteria: none

10.4 Acceptance criteria for architecture

This chapter summarizes acceptance criteria for architecture.

Note: All major approaches to Software Architecture Evaluation are qualitative evaluations, they focus on identifying risks, non-risks and architectural tradeoffs[5] with respect to explicitly stated quality goals.

There is no algorithm to evaluate or rate a given software architecture with respect to a set of quality-goals – every assessment or evaluation has to be performed manually and is based strictly on evaluators' experience.

10.4.1 Related to product quality metrics

Internal Quality The ISO-9126-3 standard defines a number of product quality metrics:

> „...defines internal metrics for quantitatively measuring external software quality in terms of characteristics and subcharacteristics defined in ISO/IEC 9126-1, and is intended to be used together with ISO/IEC 9126-1" (from ISO-9126-3, page 1)

Especially ISO/IEC 9126-1 part 8.5 (Maintainability metrics) is relevant for evaluating an architecture with respect to its maintainability, flexibility and understandability.

10.4.1.1 Program maintainability

Another useful source for software maintanability is the Software Engineering Institute *Maintainability Index for Measuring Program Maintainability*, available online[6].

The key aspect of this approach is the following:

> In this specific technology, a program's maintainability is calculated using a combination of widely-used and commonly-available measures to form a Maintainability Index (MI). The basic MI of a set of programs is a polynomial of the following form (all are based on average-per-code-module measurement):

$$MI = 171 - 5{,}2 \times \ln(\text{aveV}) - 0{,}23 \times aveV(g') - 16{,}2 \times \ln(\text{aveLOC}) + 50 \times \sin\sqrt{2{,}4 \times p}$$

The coefficients are derived from actual usage. The terms are defined as follows:

aveV *average Halstead Volume V* per module (see Halstead Complexity Measures)

aveV(g') *average extended cyclomatic complexity* per module (see Cyclomatic Complexity)

aveLOC the *average count of lines of code* (LOC) per module; and, optionally

perCM *average percent of lines of comments* per module

Oman develops the MI equation forms and their rationale [Oman 92a]; the Oman study indicates that the above metrics are good and sufficient predictors of maintainability. Oman builds further on this work using a modification of the *MI* and describing how it was calibrated for a specific large suite of industrial-use operational code [Oman 94].

---

5 See „Software Architecture Review and Assessment (SARA)", http://philippe.kruchten.com/architecture/SARAv1.pdf" or Paul Clements et al „Evaluating Software Architectures", Addison-Wesley 2003.

6 http://www.sei.cmu.edu/str/descriptions/mitmpm.html

Oman describes a prototype tool that was developed specifically to support capture and use of maintainability measures for Pascal and C [Oman 91]. The aggregate strength of this work and the underlying simplicity of the concept make the MI technique potentially very useful for operational Department of Defense (DoD) systems. [taken from the website[6]].

It is currently not validated wether such metrics can be applied or adapted to systems based upon non-procedural programming (like XSLT & Cocoon) or multi-paradigm systems.

Author's Note: ToDo: This SEI metric can also be included, at least for the java parts of the architecture. I advise NOT to make it a acceptance criterion.

10.5 Acceptance criteria for development

This paragraph summarizes acceptance criteria for development, covering the following aspects:

10.5.1 Source code artifacts

Source code artifacts are lines of source code solving a particular detail of the architecture.

10.5.2 General coding conventions

Source code shall comply to the rules given in 1.4 "'C' specific implementation standards" on page 1-7.

10.5.2.1 Criterion 3-1: (Prio A) All artifacts contain documentation header

Explanation: All development artifacts shall contain a *header comment*, briefly describing its responsibilty or task. No exception.

→ See "Source File Headers" in volume 3 page 1-7.

Verification: Inspection.

Related Criteria: None

10.5.2.2 Criterion 3-2: (Prio A/B) No duplicate code

Explanation: No duplicate code, as this leads to severe maintainability problems.

Priority is A, but due to current state of development artifacts it can be deferred.

Verification: Shall be automatically checked during daily build by static code analyzer, e.g. CheckStyle (Java) or PMD (Java, JSP, C++, PHP).

Open Issue: PMD supports several languages and includes instructions on how to add more. Two (relatively simple) interfaces have to be implemented to allow copy-paste checking of xslt-files.

Related Criteria: None

10.5.3 Related to JAVA coding conventions and styleguides

10.5.3.1 Criterion 3-3: (Prio B) Inline documentation for central classes

Explanation: Central classes shall be inline-documented to facilitate understanding.

Verification: Inspection.

Related Criteria:

10.5.3.2 Criterion 3-4: (Prio C) Uniform handling of recurring implementation aspects

Explanation: Recurring implementation aspects shall be handled uniformely to increase re-use on both conceptual and implementation level. Such aspects include

- Exception handling (shall be described in Java coding styleguide or in architecture documentation)
- Logging and tracing.
- Session handling.
- User authentication and authorization.

Verification:

- For exception-handling and logging: automatic checking (with either CheckStyle or PMD)
- Inspection.

Related Criteria: None.

10.5.3.3 Criterion 3-5: (Prio B) Use code metrics as indicator for potential risks

Explanation: Code metrics (like cyclomatic complexity or coupling) can (!) be used as indicators for potential risks within code segments.

Note: Before a proper baseline for current values can be established, a complete cleanup of existing sources should be performed (eliminiate all unused source artifacts, remove duplicate code).

Verification: Automatic measurement, should be established in daily build and monitored by development lead.

Related Criteria: None.

10.5.3.4 Criterion 3-6: (Prio C) Centralized text- and log messages

Explanation: Text and log messages could be centralized (low prio).

Verification: Sourcecode-checker can verify, that no String-constants are used in Java sourcecode.

Related Criteria:

10.5.3.5 Criterion 3-7: (Prio C) Avoid usage of side-effects in programming

Explanation: Relying on side-effects in programs causes severe maintenance und understandability risks. An example of side-effects are database triggers – which are usually completely unrelated to „conventional“ programs and are activated by a database as side-effects to „conventional“ programs manipulating database entries.

Although database triggers can manipulate relational data with very high performance, their usage in java applications is strongly discouraged, as triggers produce side effects not described/prescribed in Java programs. This can lead to severe misunderstandings or programming mistakes.

Exception: If DB-triggers are structurally needed (e.g. for performance reasons), they may be used. Under all circumstances, the usage of DB-triggers has to be made explicit within all java code manipulating the tables modified by the triggers. This holds for every kind of side-effects in programming. A list of database triggers or other side-effects-related programming plus their purpose has to be included in the documentation.

Verification: Inspection.

Related Criteria:

- Criterion 3-1: (Prio A) All artifacts contain documentation header
- Criterion 3-10: (Prio A) Keep All Required Artifacts under Version Control

10.5.4 Related to Cocoon & XSLT Coding conventions and style-guides

See sections 2.2 for further details on web application and Cocoon.

10.5.4.1 Criterion 3-8: (Prio C) Describe Responsibility of Stylesheets

Explanation: All XSL stylesheets shall contain a „responsibility“ section, briefly describing the reason, why this stylesheet exists. No exception.

Verification: Automatic verification (possibly with PMD or CheckStyle with appropriate rules).

Related Criteria:

- Criterion 1-16: (Prio A) Describe blackbox purpose, interfaces and variability
- Criterion 1-17: (Prio A) Describe code artifacts mapping of building blocks
- Criterion 3-1: (Prio A) All artifacts contain documentation header

10.5.4.2 Criterion 3-9: (Prio C) Inline-Documentation of (major) XSL stylesheets

Explanation: All stylesheets shall be documented using xsldoc

Verification: Automatic verification (possibly with PMD or CheckStyle with appropriate rules).

Related Criteria:

- Criterion 1-17: (Prio A) Describe code artifacts mapping of building blocks
- Criterion 3-1: (Prio A) All artifacts contain documentation header
- Criterion 3-3: (Prio B) Inline documentation for central classes

10.5.5 Criteria related to Version Control

10.5.5.1 Criterion 3-10: (Prio A) Keep All Required Artifacts under Version Control

Explanation: All artifacts required to build release are under version control, including the following:

- All source files
- All external libraries or tools required to build and deploy the system
- All build-files or makefiles

- All configuration files and deployment descriptions
- All testcases and test-data
- All scripts required to generate or modify the database,
- Complete documentation, including MS-Word documents, UML-models, diagrams, presentations etc..

No exception allowed. Secondary artifacts (generated classes)

Verification: Smoketest: Clean checkout into empty directory, build, run-all-tests, no build problem, no test failure.

Note: Note: A „clean-all“ in developer environment does not verify this criterion. Related Criteria:

10.5.5.2 Criterion 3-11: (Prio A) Write commit comments for every important change

Explanation: For every artifact changed for a specific release, the reason for changing is contained in the commit-comments of the change. No exception allowed.

Important changes can either be one of the following:

- bug/issue (with a reference to ticket- or bug-id).
- new requirement (with a reference to the appropriate spec, use-case or similar).
- refactoring required to optimize or streamline development & architecture. In that case, an outline, concept or fundamental idea of the refactoring has to be described.

In a series of commits by a single developer or for a single reason, not every single commit has to be commented in that sense, but at least one of several consecutive commits shall contain this information (as many commits are only performed to safely store data in the central repository).

Note: This is a project relevant aspect. Change committments are not coveredby product documentation.

Verification: Inspection.

Related Criteria: Criterion 1-4: (Prio A) ArchDoc - Release Notes & Change History

10.5.5.3 Criterion 3-12: (Prio B) Remove unused artifacts from latest/head version

Explanation: When a release is completed, the latest checkout-version (head) shall only contain files required for this release. It shall be avoided to include unneccessary files in the current releases.

Note: This is an aspect to project management. However this claim applies to product documentation as much as old chapters (documents) shall be removed from the product documentation.

Verification: Inspection

Related Criteria:

11 Installing ezRead accompanying Tools

11.1 Who should install? . . . 163
11.2 How to setup the documentation system . . . 163
11.2.1 Installation. . . . 164
11.2.2 Removal . . . 164
11.3 Manual Installation . . . 164
11.3.1 Associating stub's .STB extension manually . . . 164
11.3.2 Associating to AcroDDE.EXE . . . 171
11.3.3 Changing Adobe Reader Security Settings . . . 174
11.4 Installing with 01_InstNd.EXE . . . 175
11.5 Removing the installation . . . 175
11.6 Installing Fonts . . . 176
11.6.1 Using standard font installation. . . . 176
11.7 Set/Clear documentation environment . . . 178
11.7.1 Setting the documentation environment . . . 178
11.7.2 Clearing the documentation environment . . . 180
11.7.3 Setting the documentation environment from command line . . . 180
11.7.4 Reading the present documentation environment . . . 181

This chapter contains technical information required for the installation of the ezRead execution environment.

11.1 Who should install?

Readers and editors using MS-WORD should follow

- Installing with 01_InstNd.EXE

Editors *using Adobe FrameMaker* (typically the Editorial Team) should follow

- Installing Fonts
- Installing with 01_InstNd.EXE

and execute

- Set/Clear documentation environment

Developers might want to know about

- Setting the documentation environment from command line
- 02_AutoHotkey_L_Install.exe and other Accompanying tools

11.2 How to setup the documentation system

The ND.API is a plug-in for the Acrobat Standard/Professional version. It needs to be installed once to enhance the Acrobat for about 30 new functions, most of them related to the management of Named Destinations.

Adobe Acrobat always has a **"plug_ins" directory** in its execution path which contains the plug-ins. An Adobe plug-in is a classical DLL but with an extension "API".

Example: C:\Program Files\Adobe\Acrobat 9.0\Acrobat**plug_ins**\

Removing the plug-in

Removing the ND.API plug-in is a easy as to delete the ND.API from the **...\Acrobat\plug-ins** directory.

The installation can be done manually or by Installing with 01_InstNd.EXE.

11.2.1 Installation

→ 11.4 “Installing with 01_InstNd.EXE”.

Sometimes this does not work if the INSTND.EXE does not recognize the installation path. It will prompt the user accordingly, then a Manual Installation shall be done.

11.2.2 Removal

→ 11.5 “Removing the installation”

11.3 Manual Installation

See 12.2.5 “01_InstND.EXE” on page 12-216 to understand the function of InstND.EXE, if you don’t trust it, you can install the ND.API manually by

1. Copy the **ND.API** to the plug-ins directory

   e.g. **C:\Program Files\Adobe\Acrobat 9.0\Acrobat\plug_ins**

2. **Associating of the ".STB" extension** with Acrobat.EXE according to 11.3.1 “Associating stub’s .STB extension manually”.

3. Set the documentation environment → 11.7.1

These steps are to be done once in order to install the ND.API properly.

11.3.1 Associating stub’s .STB extension manually

To work properly, the .STB stub’s extension needs to be associated to the Acrobat Reader. Unless done through 11.2.1 “Installation” with the INSTND.EXE, a manual procedure is explained below.

11.3.1.1 Associating .stb on Windows 7 and later versions

File type association is made by selecting the properties of a file in the file explorer

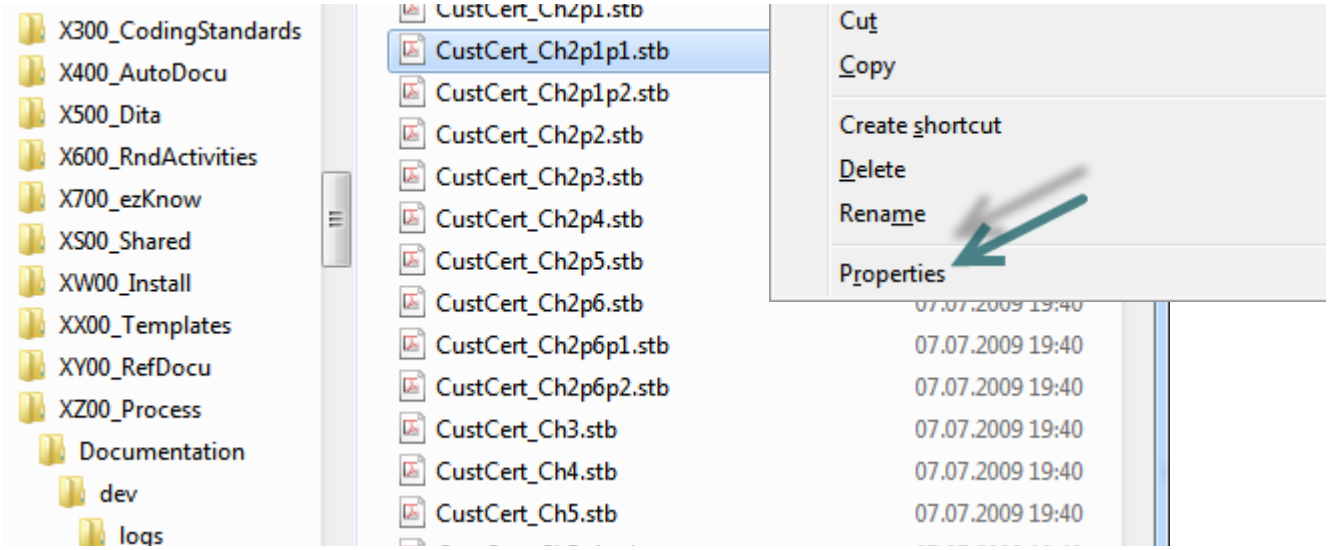


Figure 11-1. Get the properties panel with right mouse click on the file

On the properties dialog, change the association

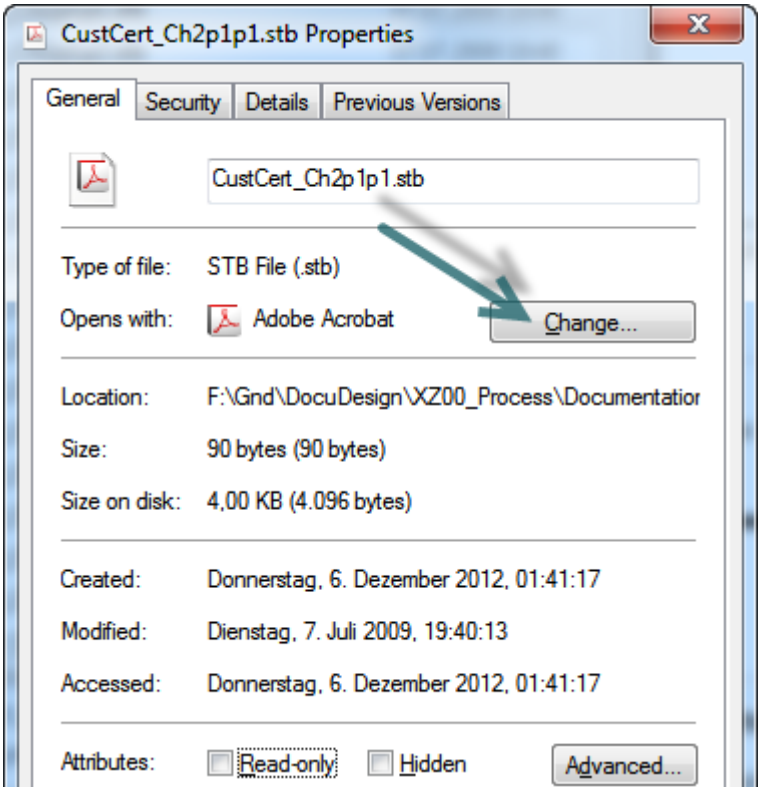


Figure 11-2. Changing the association status

Select the Adobe Acrobat as target for the STB extension. Typically it will not be recommended, so you will have to browse to Acrobat's program directory
(e.g. C:\Program Files (x86)\Adobe\Acrobat 10.0\Acrobat\Acrobat.exe)

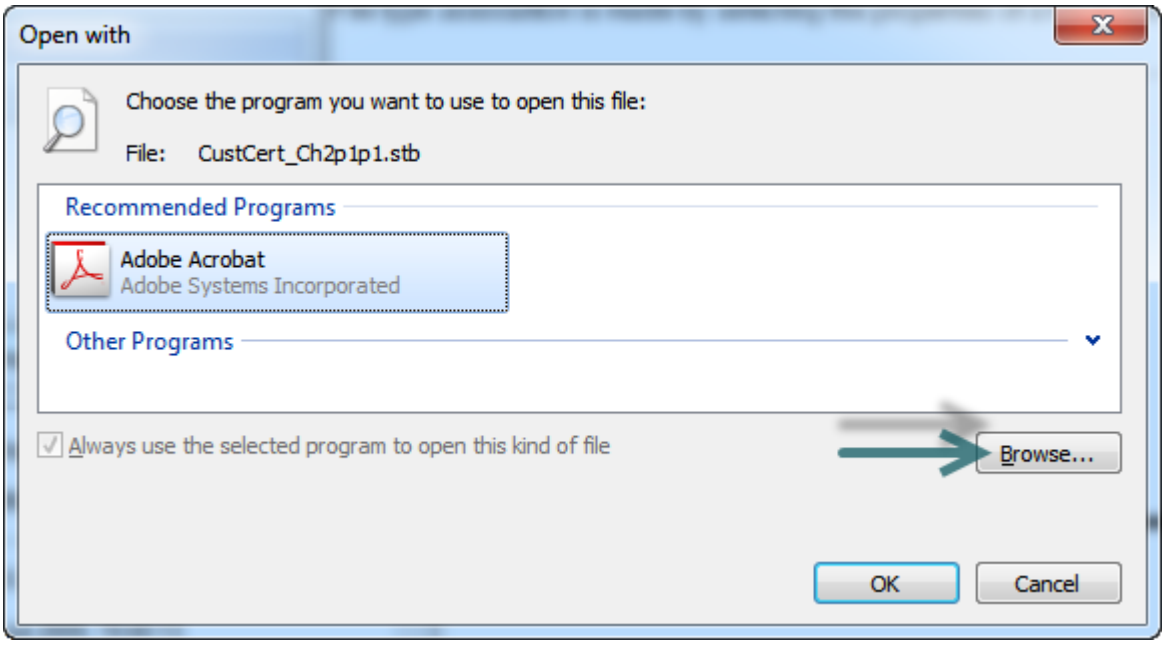


Figure 11-3. Browse for Acrobat.exe if it is not part of the recommended programs.

11.3.1.2 Associating .STB on Windows XP and earlier

Windows XP organized the associate using the **Tools - Folder Options** menu in the file explorer. The mechanism is shown here.

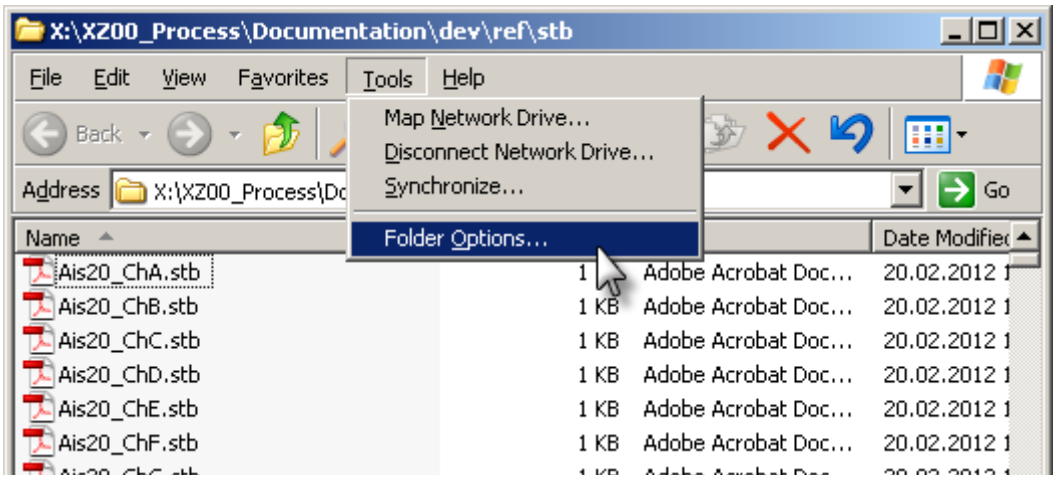


Figure 11-4. Manual file association in Windows Explorer

Create new association

To create a new association for the extension STB. Click on "New".

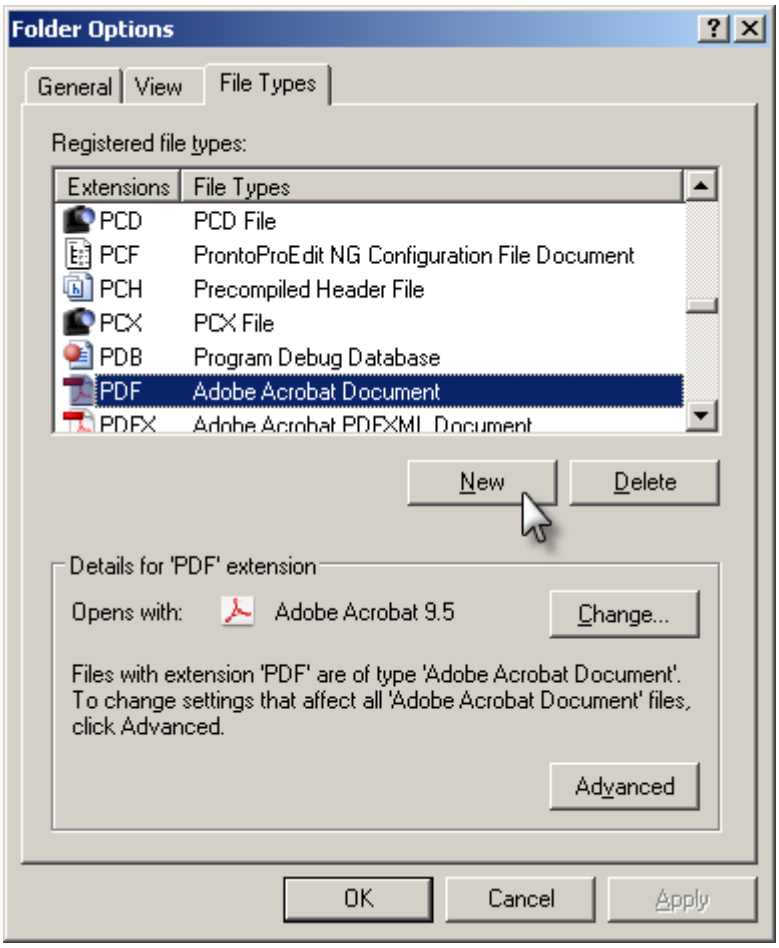


Figure 11-5. Press "New" to create new association

Enter STB

Enter the new extension "STB" for the stubs

Figure 11-6. Enter "STB" to create the stub extension association

Uncheck Warning

Unless you uncheck the warning, Windows will warn you when opening a file with the new file type. Therefore you need to uncheck the warrning clicking the "Advanced" button.

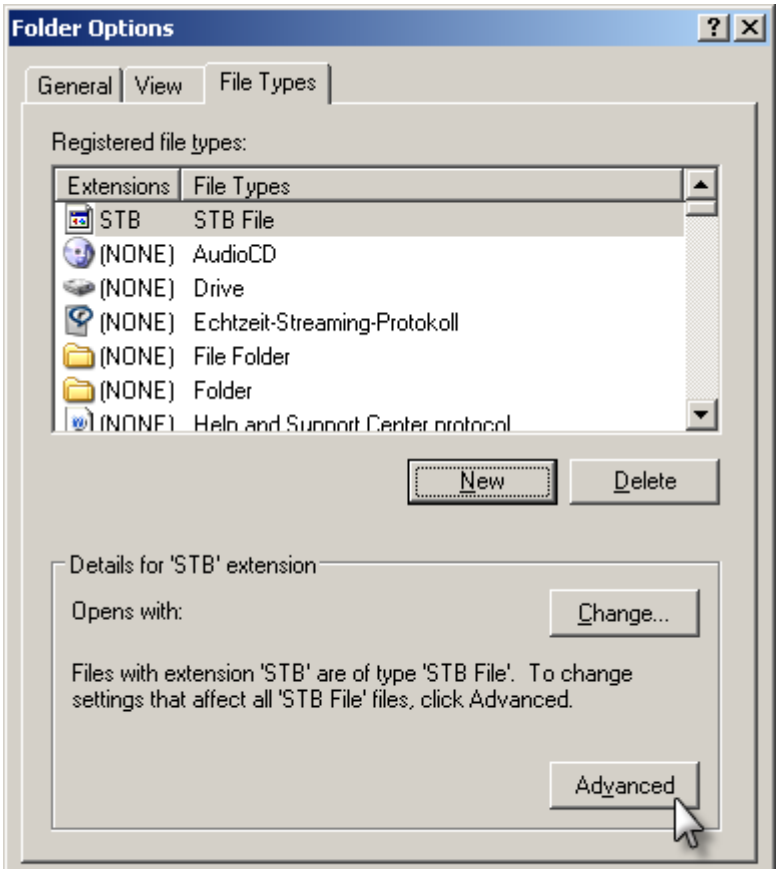


Figure 11-7. Click the Advanced key to disable warnings

Uncheck the "Confirm open after download" checkbox as shown in Figure 11-8.

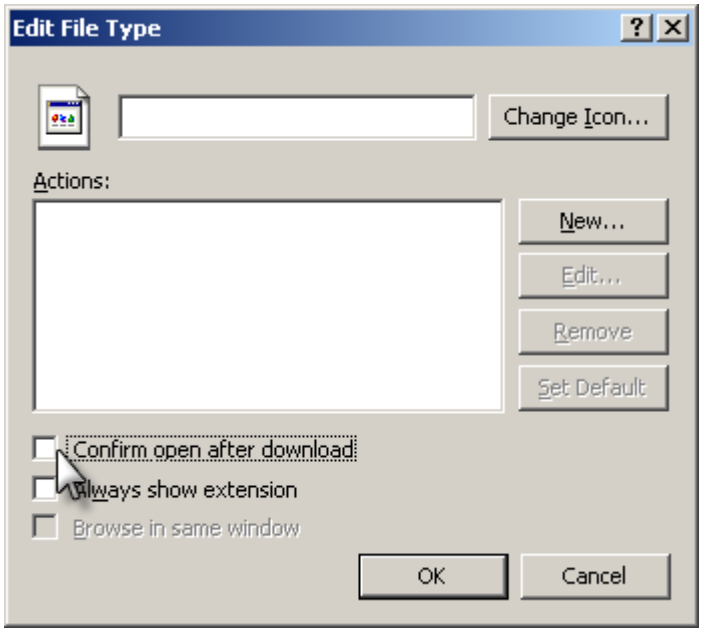


Figure 11-8. Uncheck the warning "Confirm open after download" → OK

Change association

Change the association of the new file type using the "Change" button

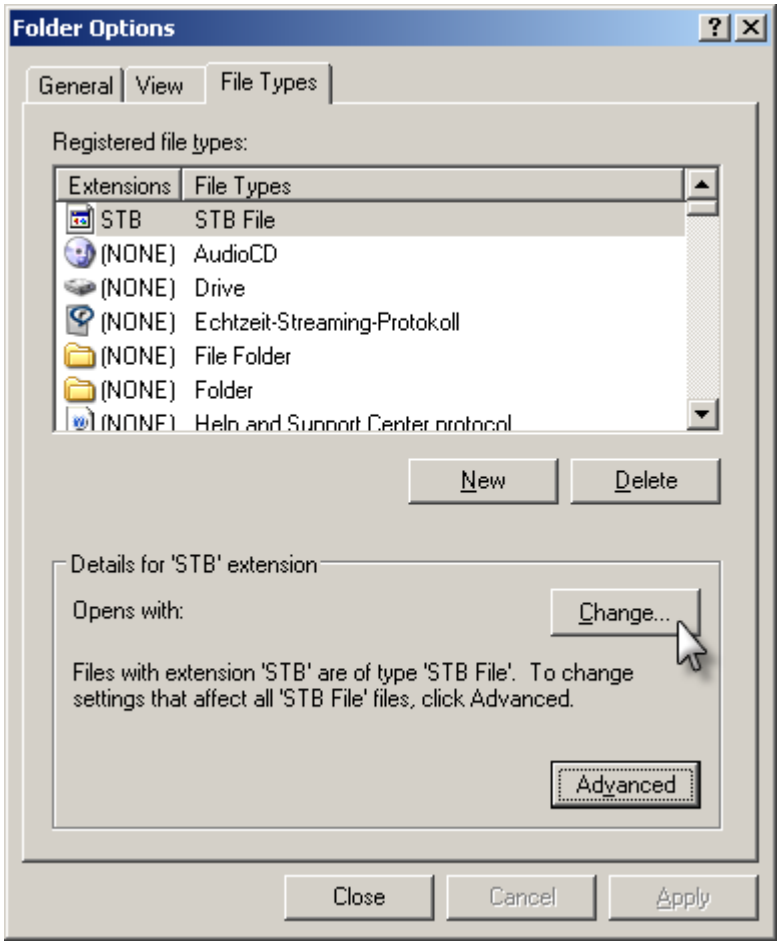


Figure 11-9. Create the association with "Change ..."

Use "Select the program from a list to determine the associated target program

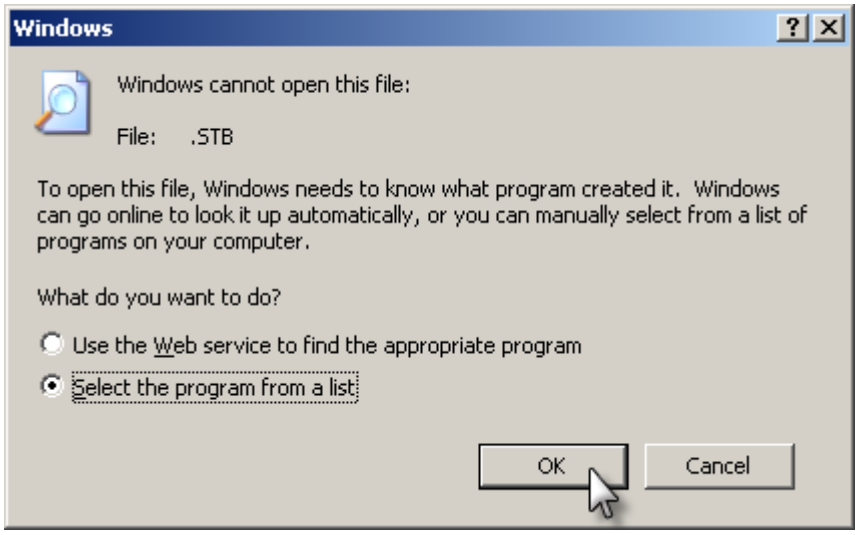


Figure 11-10. Select the program from a list

Find the Adobe Acrobat entry to associate the extension ".STB". If you cannot find this entry in either "`Recommended Programs`" nor "`Other Programs`", then use the "Browse"-key to select Acrobat.EXE from its diectory

(e.g. C:\Program Files\Adobe\Acrobat 9.0\Acrobat\plug_ins\).

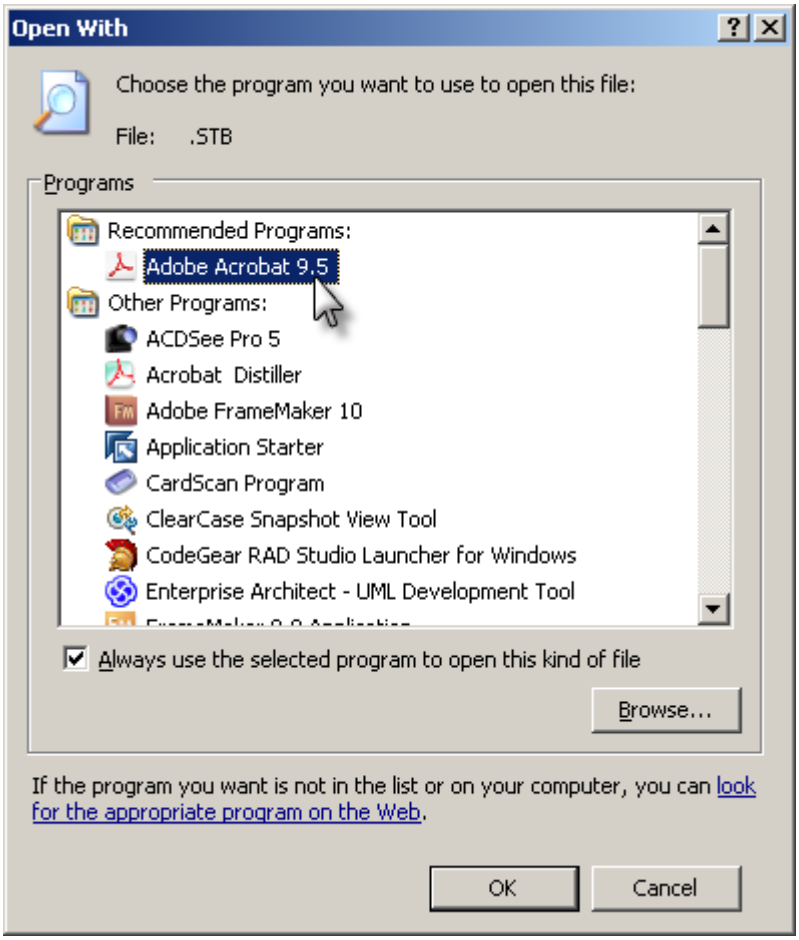


Figure 11-11. Associate to Acrobat

If you do not own Adobe Acrobat, go to 11.3.2 “Associating to AcroDDE.EXE”.

You’re done - the icon in the STB file type should now indicate the Adobe Acrobat. Press "Close" to finish.

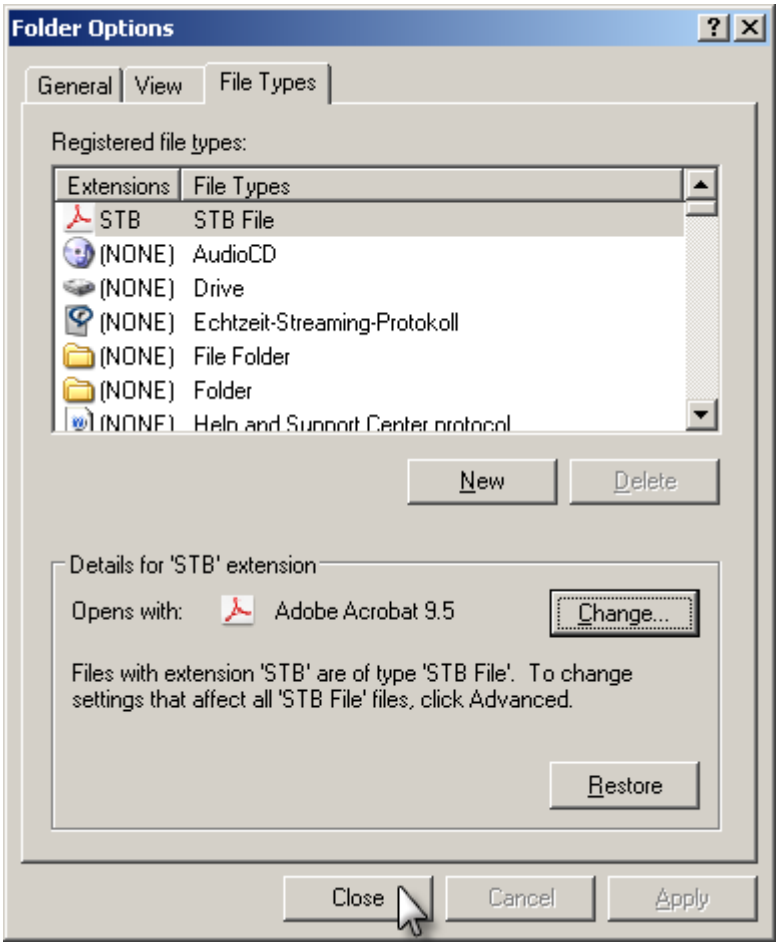


Figure 11-12. Close the options

11.3.2 Associating to AcroDDE.EXE

If you do not have the Acrobat version installed but using the Adobe reader, the file association STB needs to be done to AcroDDE.EXE. This will allow to navigate with the free Adobe Reader. The AcroDDE.EXE interpretes the stub and uses the Dynamic Data Exchange (DDE) interface of the Adobe Reader to control the invokation and page/chapter selection. The ND.API plug-in cannot be installed in Adobe Reader.

The DDE interface is described in [AdobeDDE#3] and [AdobeAPI#2].

To associate to the AcroDDE.EXE follow the steps above until Figure 11-11. Then continue with the following steps.

Use "Browse"

The "Browse" button will open a file selection window to select the AcroDDE.EXE. Of course you should know where you placed it after download.

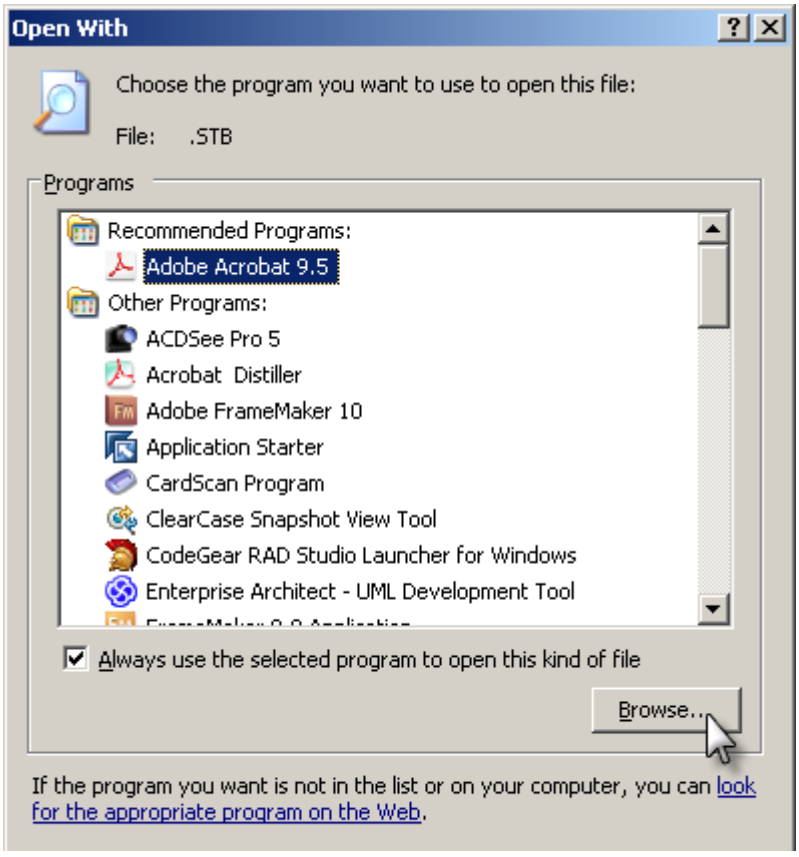


Figure 11-13. Browse for association

Find the location of the AcroDDE.EXE and select it and click on the button "Open"

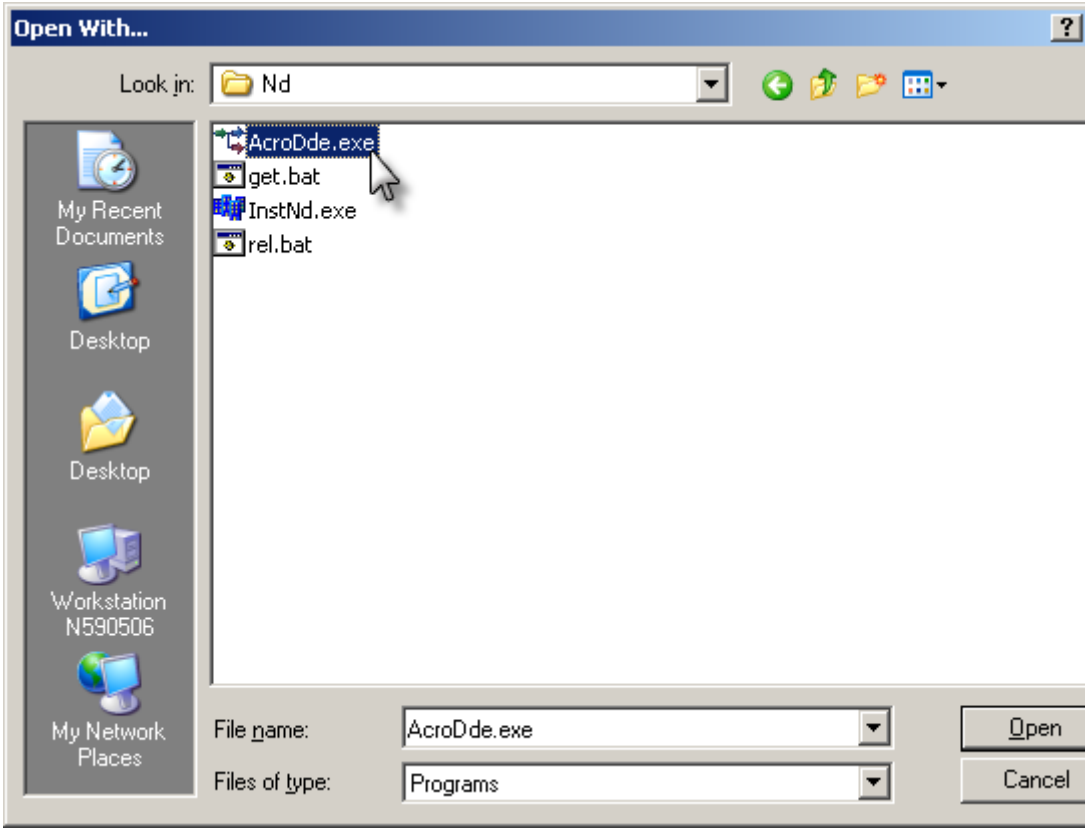


Figure 11-14. Associate to AcroDde.exe

AcroDDE.EXE will appear in a selection list now. Select it and press "OK".

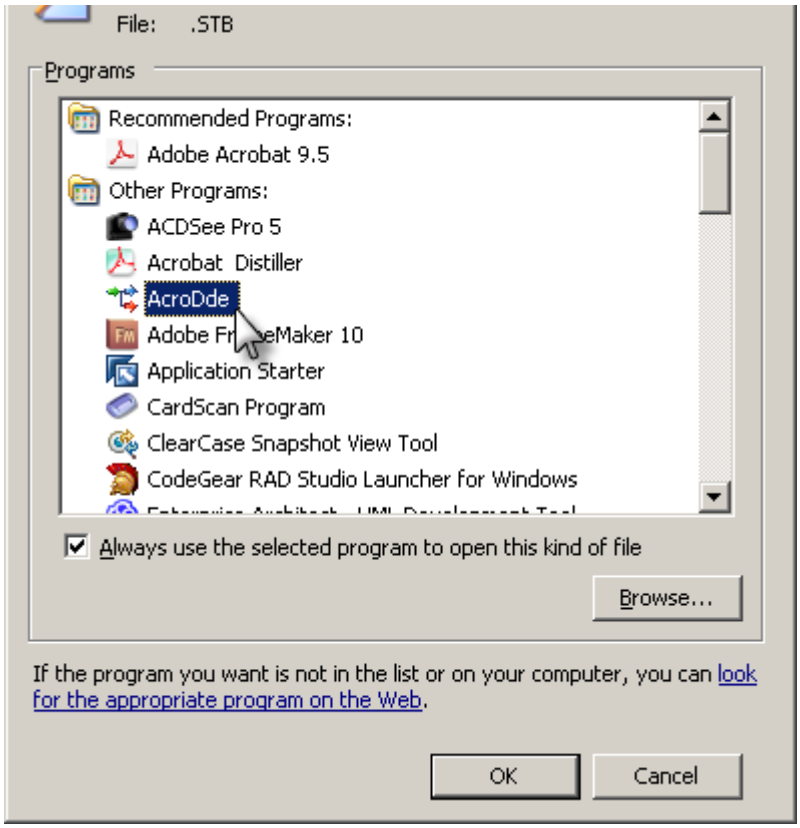


Figure 11-15. Associate to AcroDde.EXE

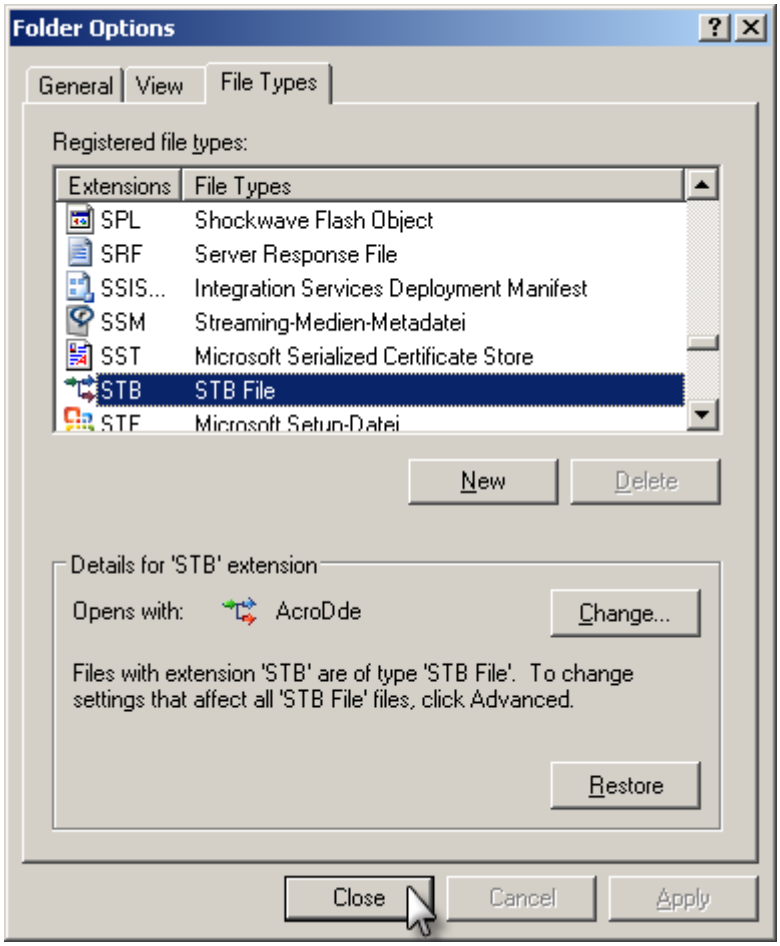


Figure 11-16. Close the Folder options

11.3.3 Changing Adobe Reader Security Settings

Trying to invoke the Adobe Reader from an STB association technically requires to contact the internal DDE server of the Adobe Reader. However this server is only started and available if the security settings of the Adobe Reader are configured appropriately.

Starting a stub file (.STB) might invoke the message

Figure 11-17. Missing DDE server in Adobe Reader

In this case start the Adobe Reader manually and goto EDIT – PREFERENCES also available with the ShortKey CTRL-K. You need to **uncheck the enhanced security settings** in the security panel.

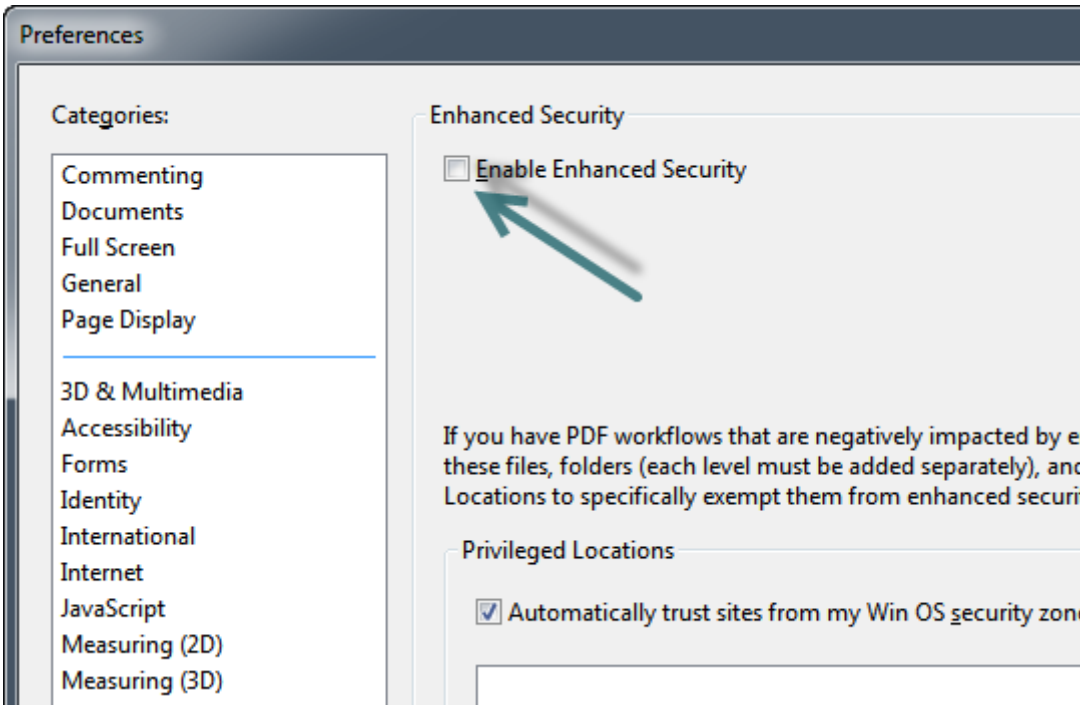


Figure 11-18. Uncheck the enhanced security settings

This will allow the Adobe Reader to start the DDE server. Hence AcroDDE.EXE is able to use this interface to navigate to the chapters.

11.4 Installing with 01_InstNd.EXE

The installation of the ND.API plug-in or its Adobe Reader equivalent AcroDDE is very easy.. The user only needs to click on the InstND.EXE in the installation directory.

All the rest is performed automatically.

What happens on installation

To better understand, what actually happens on clicking InstND.EXE refer to 12.2.5 “01_InstND.EXE” on page 12-216.

11.5 Removing the installation

If the **InstNd.EXE** is invoked with the option “**/r**”, then the present installation of the ND.API plug-in or the AcroDDE is removed. The recommended use is to start **InstNd.EXE /r** from the command line prompt. To run from a shortcut, create a shortcut to the InstNd.EXE and modify the *properties* as shown below.

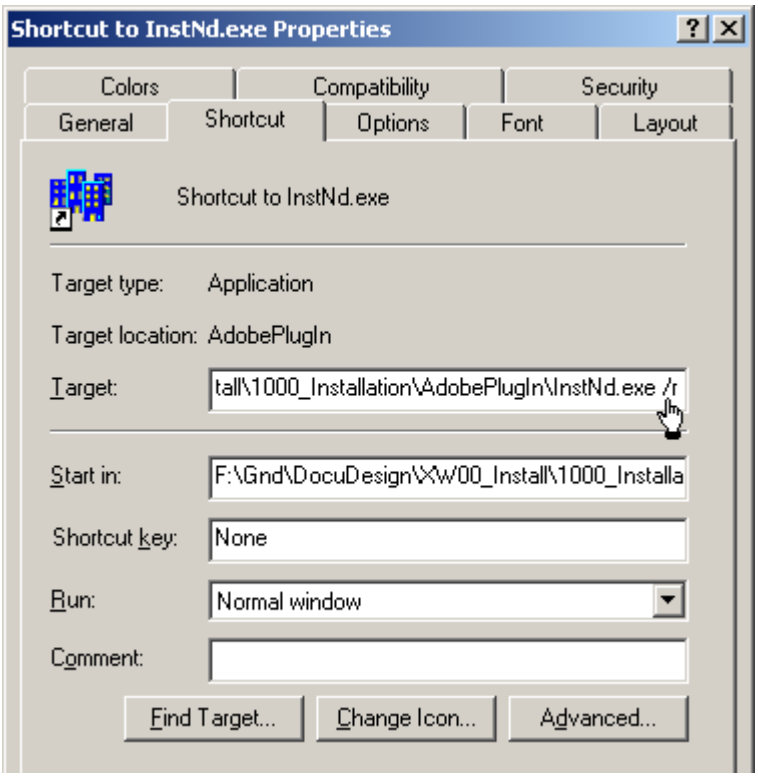


Figure 11-19. .ND.API work directory tree

11.6 Installing Fonts

This chapter is only relevant for the use of AdobeFramemaker and the particular fonts required to comply with IBM BookMaster technology.

In order to use the full capabilities of the FrameMaker documentation it is necessary to install several fonts. The fonts are available in the *..\1000_Installation\Fonts*

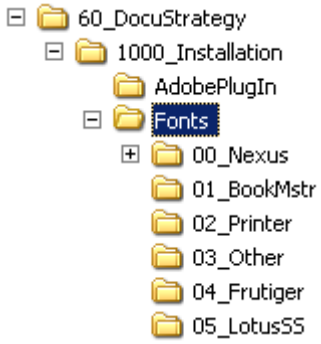


Figure 11-20. Fonts directory

Windows 7 and later allows to copy the Font Files directly to the Fonts directory. The mechanism is currently not explained in this document.

11.6.1 Using standard font installation

The standard installation can be necessary if it is not possible (technical or organizational reasons) to use the Nexus installation. In the Tools *../Fonts* folder is available which contains five sub-folders.

Windows 7 and later will require a direct copy, the Install function will no more be available in the Fonts folder.

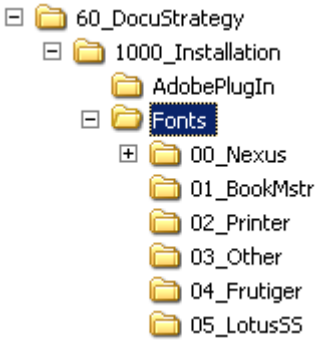


Figure 11-21. Fonts directory

Install the Fonts using Start - *Settings - Control Panel -* Fonts. in the suggested order *01..03*. The other fonts *04..05* are optional.

> The fonts *04_Frutiger* is only required if documents are to be created with the G&D internal Frutiger font. Currently, none of the paragraph styles used in FrameMaker uses this font.
>
> The font *05_LotusSS* is for compatibility with Lotus applications (e.g. Nimrod Font). This font is optional

When the Fonts Folder is opened, then use *File - Install - New Font ...* and you see

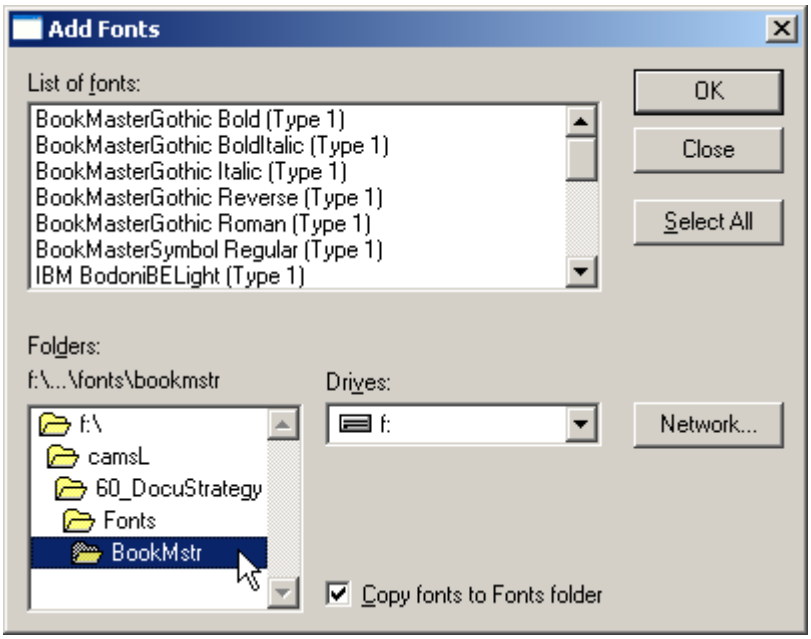


Figure 11-22. Fonts directory

after you selected the drive and directory where your documentation is located.

Use *Select All - OK* in order to install the *BookMstr* fonts. Then proceed similar with the *Printer* and *Other* fonts.

If you receive a panel indicating that the fonts are already installed

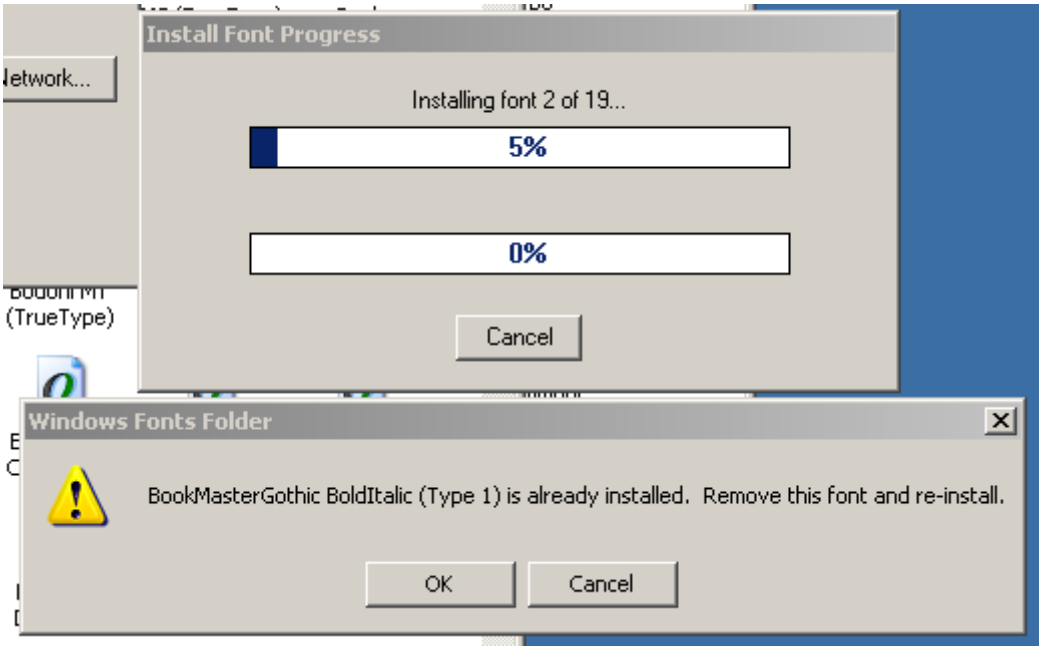


Figure 11-23. Fonts warning

then press ***OK*** to continue installation.

11.7 Set/Clear documentation environment

The ND.API plug-in needs a work directory in order to save exported files. In the example below the work directory is `C:\ProgramData\ezRead`

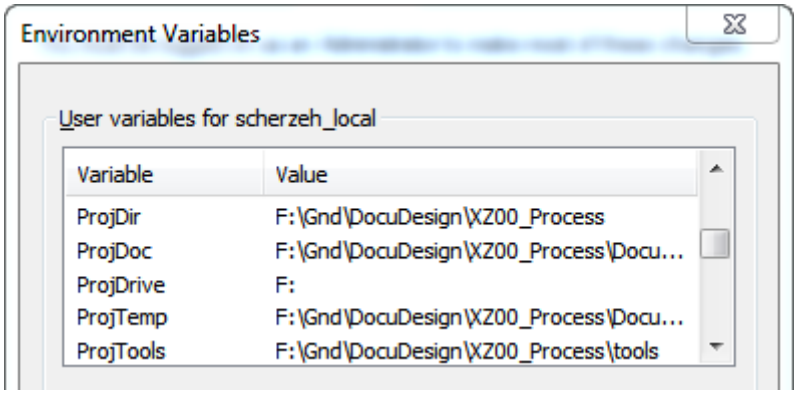


Figure 11-24. .ND.API work directory tree

The ND.API allows to set this working directory using the "*Set documentation environment*" function in the "*Advanced - Manage Destinations*" menu of the *Adobe Acrobat Reader*. This function can be manually chosen, but will also be automatically invoked, if the ND.API does not find an environment on trying to import/export items.

Present documentation environment

The present documentation environment path may read from the "About ND-Plugin" function of the Menu.

11.7.1 Setting the documentation environment

Setting the environment will automatically execute three processes

- Create directory tree
- Set environment (persistently)
- Append PATH

Create directory tree: To set the environment select *Set documentation environment*". The following panel appears

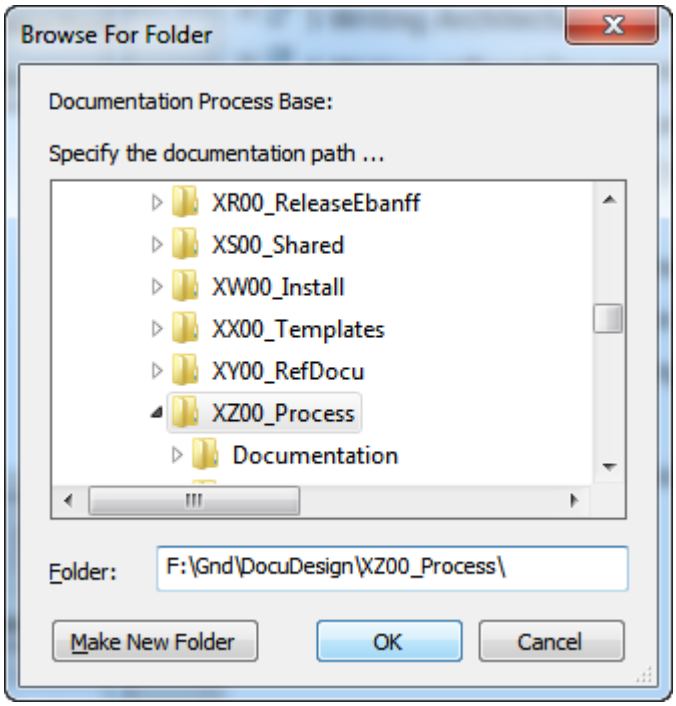


Figure 11-25. Select Documentation environment

Choose (or make) a working folder - preferably in you project tree. In this example we will choose *XZ00_Process*. After confirmation the following directory structure will automatically be created. The "Documentation" directory shall note be selected if already present. Instead it will be created automatically with the confirmation.

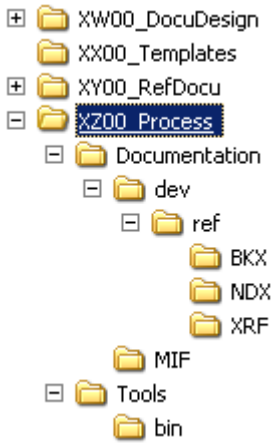


Figure 11-26. Generated documentation environment tree under XZ00_Process

BKX/NDX XRF/STB directories

Three directories Documentation – BKX (Bookmarks) – NDX (Named Destinations) and XRF (Cross References) are generated. BKX and NDX are used to store bookmarks and named destinations. The XRF directory is used for storing references being generated by auto-documentation tools. The nesting chain *Documentation-dev-ref* is intentionally chosen to resolve relative pathes when using auto-documentation features.

The MIF directory is also provided for auto-documentation to store intermediate MIF files being generated from auto-documentation tools.Writing documentation it is not required but may serve as a user workdir.

The *Tools\bin directory* is created to store project specific tools. This is an interesting aspect because the PATH variable in the environments setting is also enhanced to seek executable files in this path.

Set environment (persistently): The function also creates or updates the following reserved user environment variables.

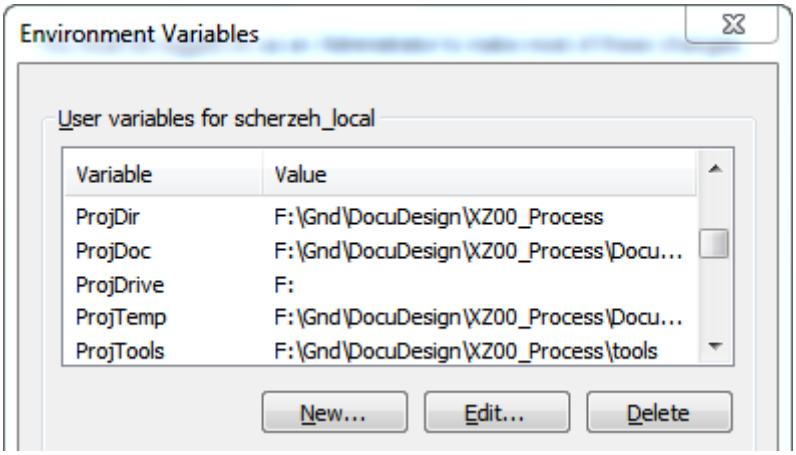


Figure 11-27. Environment variables set for working environment

These user variables are entered in the registry, therefore they will stay until the environment is cleared or another environment is set. A broadcast message is launched to all Windows programs in order to update their environment. Therefore the variables are available instantly without restarting the PC.

Append PATH: For use with Accompanying tools, the PATH environment variable is extended to the content of *%ProjTools%\bin*. %ProjTools% is that variable which points to the Tools directory (→ Figure 11-26). The actual executable files are expected to reside in *%ProjTools%\bin*. A use case for using command line tools is found in 7.1.5.3 “Case 3: No bookmarks in PDF, but Table of Contents” on page 7-94.

11.7.2 Clearing the documentation environment

Advanced - Manage Destinations - Clear documentation environment in Acrobat Reader clears the user variables from the registry and the PATH statement. It does not remove the directories created under Create directory tree.

The function is seldom used but implemented for proper housekeeping. It is however more often the case, that the documentation environment changes, typically, when the user changes between two projects.

11.7.3 Setting the documentation environment from command line

Developers might have a need to set the documentation environment from a command line, often used from batch files. This can be done using the SetEnvReg.EXE described in 12.2.3 “SetEnvReg.EXE” on page 12-214.

11.7.4 Reading the present documentation environment

The present documentation environment path may read from the "About ND-Plugin" function. ("About th ND-Plugin").

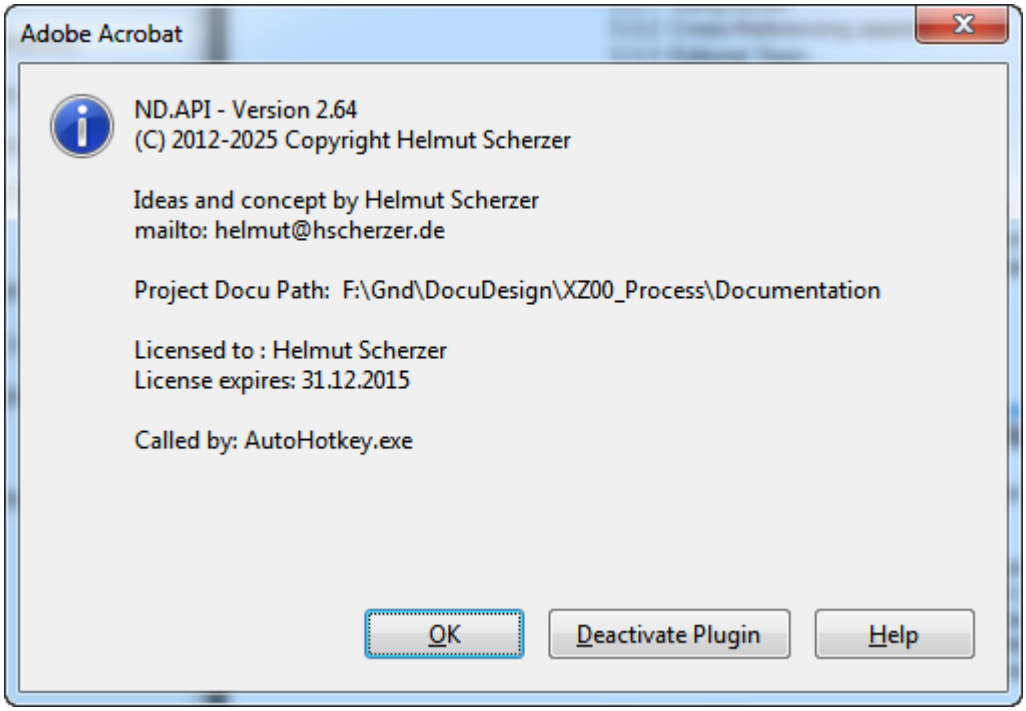


Figure 11-28. Indicating current environment and caller by "About ND"

The panel shows the current project environment and also some license information.

The "called by" field helps to understand who actually called the Acrobat.EXE. This can be helpful for maintenance, but in particular the information is used inside the ND plug-in to avoid its own invokation if Acrobat is used from an Internet Explorer PDF view.

Since a browser does not have the right to access files or the registry, an appropriate error message about a missing serial number would be annoying to the reader.

12 Technical Guide

12.1 The ND.API plug-in 183
12.1.1 Navigate Documents 184
12.1.2 Set Named Destination 185
12.1.3 Export/Import named destinations 185
12.1.4 NDX:Named destinations - CSV File structure 185
12.1.5 Clear named destinations 187
12.1.6 Generate named destinations from bookmarks 188
12.1.7 Generate bookmarks from named destinations 188
12.1.8 Generate named destinations/bookmarks from comments 191
12.1.9 Get named Destination from selected Text range 194
12.1.10 Export/Import Bookmarks 194
12.1.11 Enumerate bookmarks 196
12.1.12 Focus selected Bookmarks 196
12.1.13 Generating stubs from Bookmarks 197
12.1.14 Generating Stubs from named destinations 198
12.1.15 Generate Stubs from Dest with coordinates 198
12.1.16 Setting Bookmark Stubs manually 199
12.1.17 Bulk generation of stubs from document title 199
12.1.18 Stub management 200
12.1.19 Create Hyperlink from selected text 200
12.1.20 Convert links 204
12.1.21 Set/Clear Documentation Evironment 206
12.1.22 File/Path Function 206
12.1.23 Set ND view reference 207
12.1.24 Fix current Zoom 208
12.1.25 Generate Slides 208
12.1.26 Exporting linked files 209
12.1.27 Clearing export directory 209
12.1.28 Remove Personalization 209

12.2 Accompanying tools 214
12.2.1 Toc2Bkx.EXE 214
12.2.2 Bkx2Stb.EXE 214
12.2.3 SetEnvReg.EXE 214
12.2.4 AcroDDE.EXE 214
12.2.5 01_InstND.EXE 216
12.2.6 02_AutoHotkey_L_Install.exe 216
12.2.7 ClipText.EXE 218
12.2.8 ClipJoin.EXE 218

12.1 The ND.API plug-in

The ND.API (“NamedDestinations.API”) plug-in adds several functions to *Adobe Acrobat Reader*. The installation is described in “Installing with 01_InstNd.EXE”.

The ND.API plug-in enhances the functionality of the Adobe Acrobat Reader by functions for named destinations and bookmarks.

- Navigate Documents
- Set Named Destination
- Export/Import named destinations
- Clear named destinations
- Generate named destinations from bookmarks

- Generate named destinations/bookmarks from comments

- Export/Import Bookmarks
- Enumerate bookmarks

- Setting Bookmark Stubs manually
- Generating stubs from Bookmarks
- Generating Stubs from named destinations
- Bulk generation of stubs from document title

- “Set/Clear documentation environment”
- Convert PDF to STB (FrameMaker)
- File/Path Function
- Set ND view reference
- Focus selected Bookmarks
- Fix current Zoom
- Exporting linked files
- Clearing export directory
- Convert links

All the functions are found in the menu under **Advanced - Manage Destinations**

12.1.1 Navigate Documents

Since Acrobat X and later did not support parent document navigation, the ND.API provides even a more powerful function of parent/child navigation.

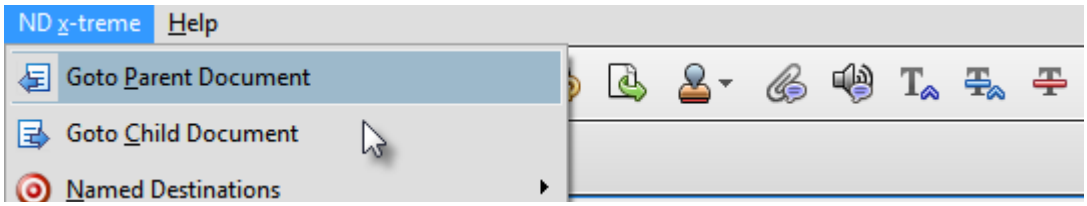


Figure 12-1. Direct Parent Child Navigation

These functions navigate through the last recent parent–child chain, even if documents in the chain are already closed – which will cause them to be reopened. The "call chaing" is initialized when opening the Acrobat Reader. Closing the reader will delete the history log.

An associated HotKey script can assign these key strokes to the intuitive keys **SHIFT + PgUp** and **SHIFT + PgDown**. Find more in 12.2.6 “02_AutoHotkey_L_Install.exe”.

```
#IfWinActive ahk_class AcrobatSDIWindow
;---------------------------
; Shift PgUp - goto previous document
;---------------------------
+PgUp::
Send !xp
return

;---------------------------
; Shift PgUp - goto previous document
;---------------------------
+PgDn::
Send !xc
return
#IfWinActive
```

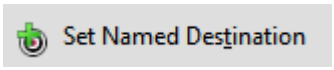
Figure 12-2. HotKey Script to jump to previous/next document by Shift + Page commands

12.1.2 Set Named Destination

As explained in 7.1.4 “How to create a named destination manually” on page 7-88, named destinations should have the prefix “M8.newlink.”. To ease this the “Set Named Destination” function was added although named destinations may be added by the native Acrobat Reader too.

There is another important advantage when using the plug-in function.

12.1.2.1 Position correction



Acrobat design flaw on creation of named destinations

If you want to add a named destinations at the rather bottom of a page, you will have to move the desired location to the top frame. However, Acrobat will always display that page in the page indicator 

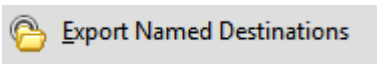
which occupies the **majority** of the window. If you align the bottom of page 9 to the top frame, then it page 10 will occupy most of the window and Acrobat will display “page 10”.

Unfortunately Acrobat will then also add a named destination at page 10 - regardless from your correct positioning. Although this can be fixed by adjusting the window size to only display page 9 - the comfort of such action is very low.

Solution by ND.API

Using plug-in function “Set Named Destination” avoids this problem by recomputing the correct position from internal information.

12.1.3 Export/Import named destinations



Preparing books for the final release can be exported and imported. If named destinations are manually added to a document (→ 7.1.4 “How to create a named destination manually” on page 7-88) then the precious work can be exported into a special BKX directory. This directory is automatically created using the “Set/Clear documentation environment” functions.

Export import of named destinations can also be helpful in the future. If a PDF document gets a new version, typically there are minor changes and lots of named destinations could be taken in - but in another place. Future migration tools will use the old and the new PDF file in order to compare changes and adjust the named destinations to the new document. Import/Export functions are essential to that development.

If named destinations are only made from bookmarks (→ 7.1.5 “How to create named destinations automatically” on page 7-92) then version update is possible simply by re-generating the named destinations from the new bookmarks. That does not prevent a wrong hit of a cross-reference to a chapter number that has changed from the update.

12.1.4 NDX:Named destinations - CSV File structure

Exported named destinations are stored in files with the extension .NDX

First record: Named destinations are exported into a comma-separated-value (CSV) file. The first record is a descriptor of the file

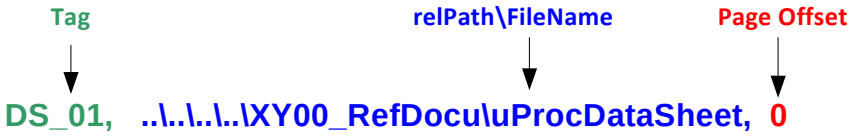


Figure 12-3. .NDX - Named destination CSV - file descriptor: first record

The **Tag** contains an *identifier* for the document. Although it does not technically hurt, for use with other tools (e.g. Auto-Documentation Tools) this identifier should be unique throughout the entire project.

The **relPath\FileName** contains the location of the file *relative to the NDX directory* which stores the CSV files (NDX). The NDX directory is created using the "Set/Clear documentation environment" function.

The **PageOffset** defines a page offset if an NDX files is used for import. On Export, the page offset is always zero.

On import, it can happen, that the location of named destinations has been determined regarding the relative coordinates on a page. The page itself, however might be wrong. This typically happens, if page numbers are retrieved from a *Table of Contents* (→ 7.1.5.3 "Case 3: No bookmarks in PDF, but Table of Contents" on page 7-94). If page numbering doesn't start with page #1 at the first page in the PDF file, then it can be necessary to indicate this offset in the NDX file to be imported.

The **PageOffset** solves this problem by allowing an offset value indicating how many pages proceed the designated page 1.

Following records: The following records of the NDX file are all of the same structure. Every line indicates a new named destination entry.

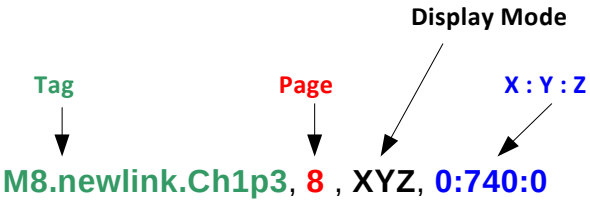


Figure 12-4. .NDX - Named destination CSV record

The **Tag** indicates the full name of the named destination. The prefix *M8.newlink* is required by Adobe FrameMaker. Whenever you insert a hyperlink to a named destination in an Adobe FrameMaker document, then FrameMaker will add the prefix "M8.newlink." ahead of the desired name. Creating a named destination "MyDestination" in a FrameMaker document will actually create a hyperlink to the named destination "M8.newlink.MyDestination".

See 7.1.4.1 "Step 1: Choose the right destination name" on page 7-88 for details for the correct naming of named destinations.

The **Page** indicates the page number in the PDF file. This can be different from the page numbering printed on the bottom of the page.

The **Display Mode** determines the way the Acrobat reader will display the page when it jumps to the named destination. Typically *FitH* or *XYZ* are the recommended values.

XYZ indicates to display the page with the named destination at the top of the page. The view will not be changed from any previous view. In the example above the value 740 determines an offset. The offset is zero-based and the actual number is an Adobe PDF internal number.

FitH will set the view of the Acrobat reader to *horizontally fit the page* into the left/ right margin. As with XYZ, the named destination will be placed at the top of the page.

Other options can be found in the “PDF Open Parameters” documentation.

The **X:Y:Z** component carries the (vertical) y-position of the named destination. The center point is the bottom left point of a PDF page. The actual numbering is an Adobe internal numbering, described in the *Adobe PDF Guide* as part of the freely available *Acrobat Software Development Toolkit*.. (see www.adobe.com/devnet/).

For our use we do not *manually* modify the generated y-component. The other components `X,Z` are not used with named destinations.

12.1.5 Clear named destinations

Clearing named destinations becomes important if *M8.newlink* named destinations shall be replaced (→ 12.1.3 “Export/Import named destinations” on page 12-185). This often happens when modified versions of a document are available and the named destinations are to be updated.

The function allows to clear named destinations. While this can be typically be done interactively by the named destination panel (*View - Navigation Panels - Destinations*) the Advanced function allows a filtering such that only *M8.newlink destinations* get cleared.

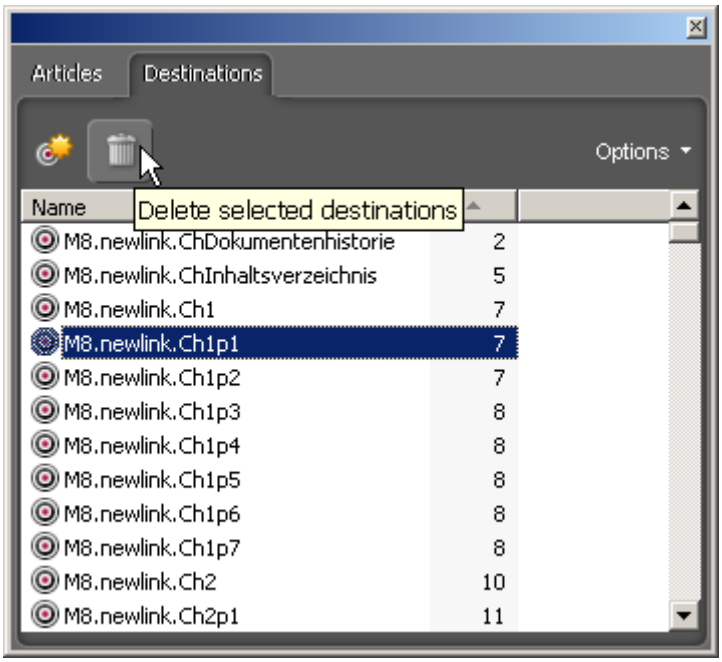


Figure 12-5. Deleting named destinations manually

Using the “*Clear named destinations function* “shows a panel

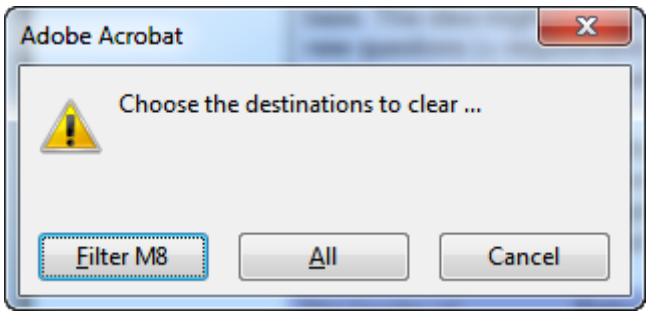


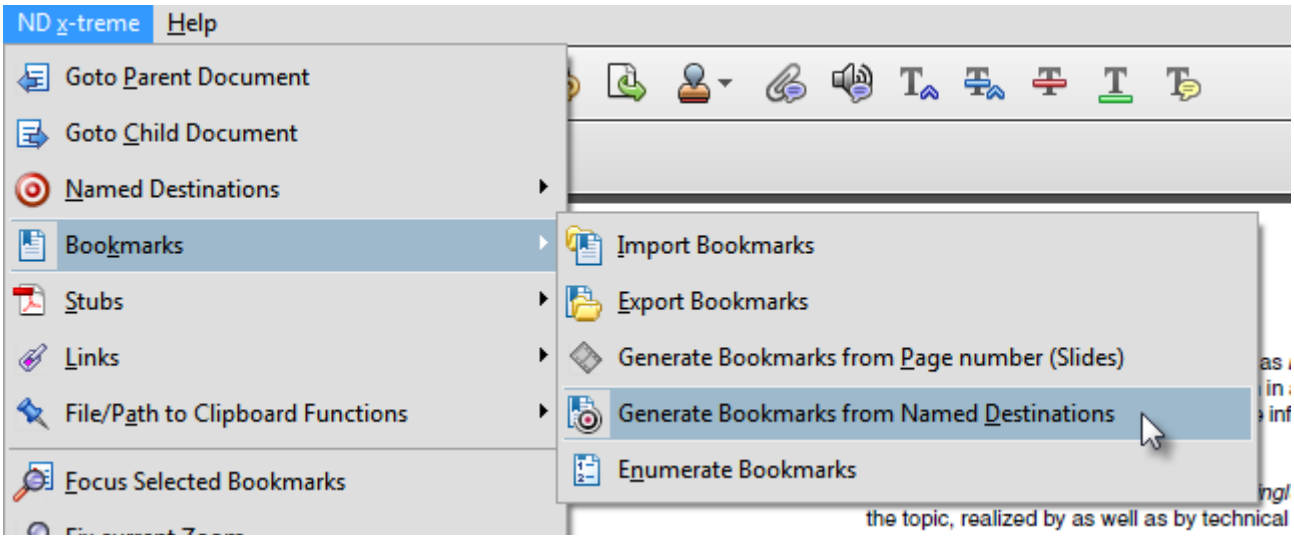
Figure 12-6. Clearing only M8.newlink destinations

Typically this filter function should be used in order to leave other named destinations untouched.

12.1.6 Generate named destinations from bookmarks

This function allows the generation of *M8.newlink* named destinations from existing bookmarks. The typical use case is described in 7.1.5.1 “Case 1: Enumerated bookmarks available in PDF” on page 7-93).

12.1.7 Generate bookmarks from named destinations



This function allows the opposite way – use named destinations to generate bookmarks. It is seldomly used for official documents and specifications which typically do contain bookmarks and if they don’t, the probability that they will contain associated named destinations – is low.

However, the case is relevant if your own document is generated by some (poor) PDF creator which creates named destinations, but lacks the proper bookmark creation. The case is also interesting if a document has strange named destinations and you want to improve.

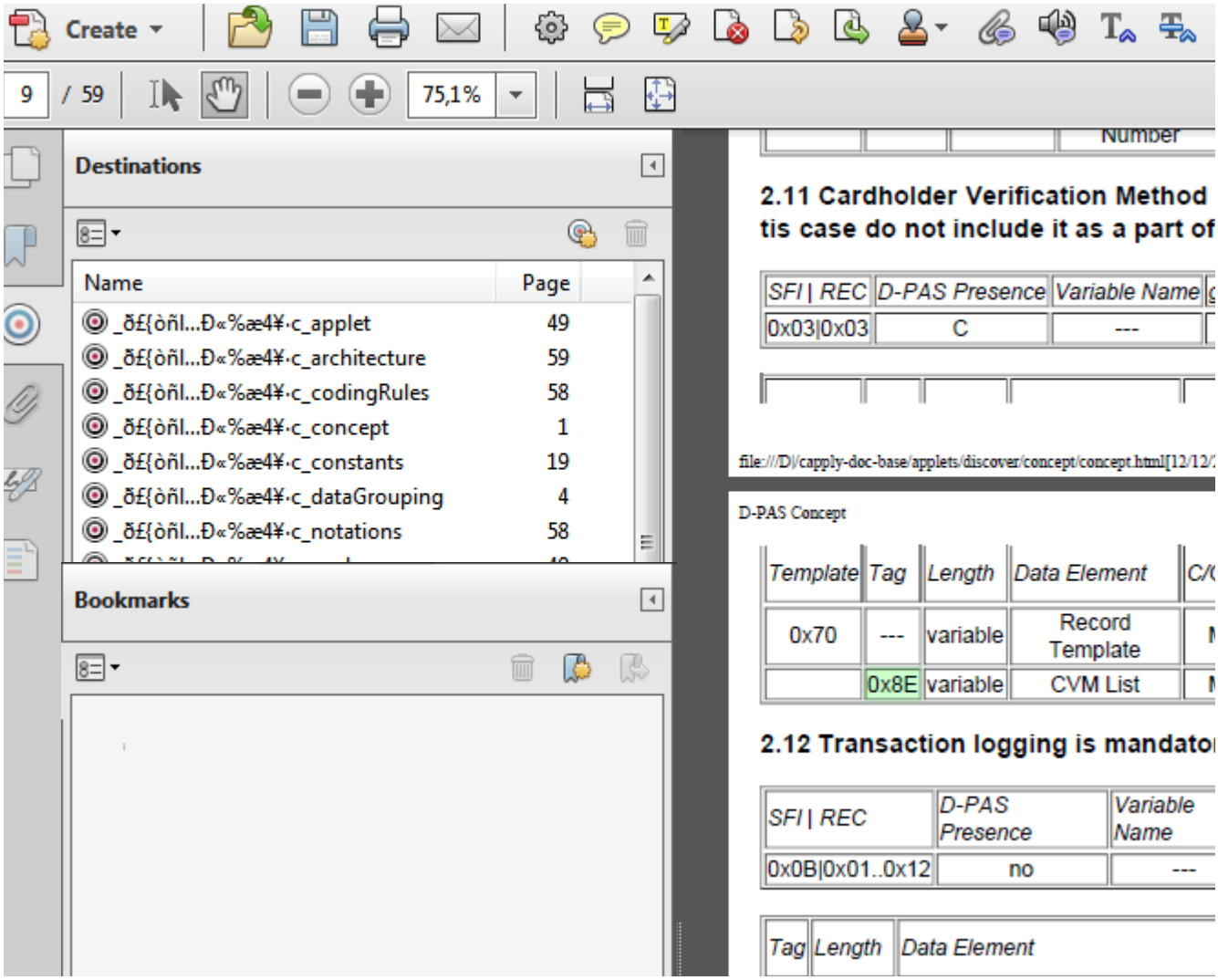
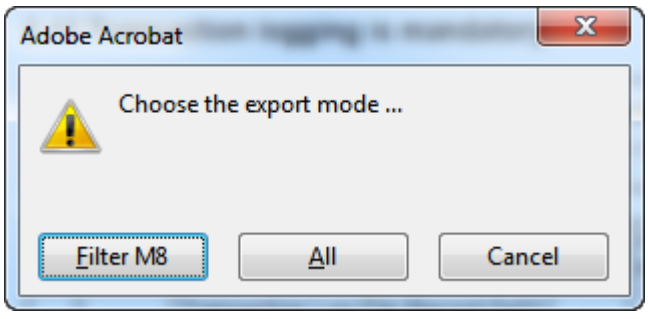


Figure 12-7. Document without bookmarks and with poor named destinations

Using - *Special Functions* - **Generate Bookmarks from named destination** shows a dialog to ask what type of named destinations shall be considered for target



For the example there is no M8.newlink... entry, therefore the "All" selection is appropriate.

The function then creates bookmarks by visiting the targets and catching the text content in the line that is closest to the target. If those are headlines, there is a good chance to successfully create bookmarks. The system also scans the potential ordering scheme, e.g. the chapter numbering and it computes the associated nesting structure

The result contains the created bookmarks

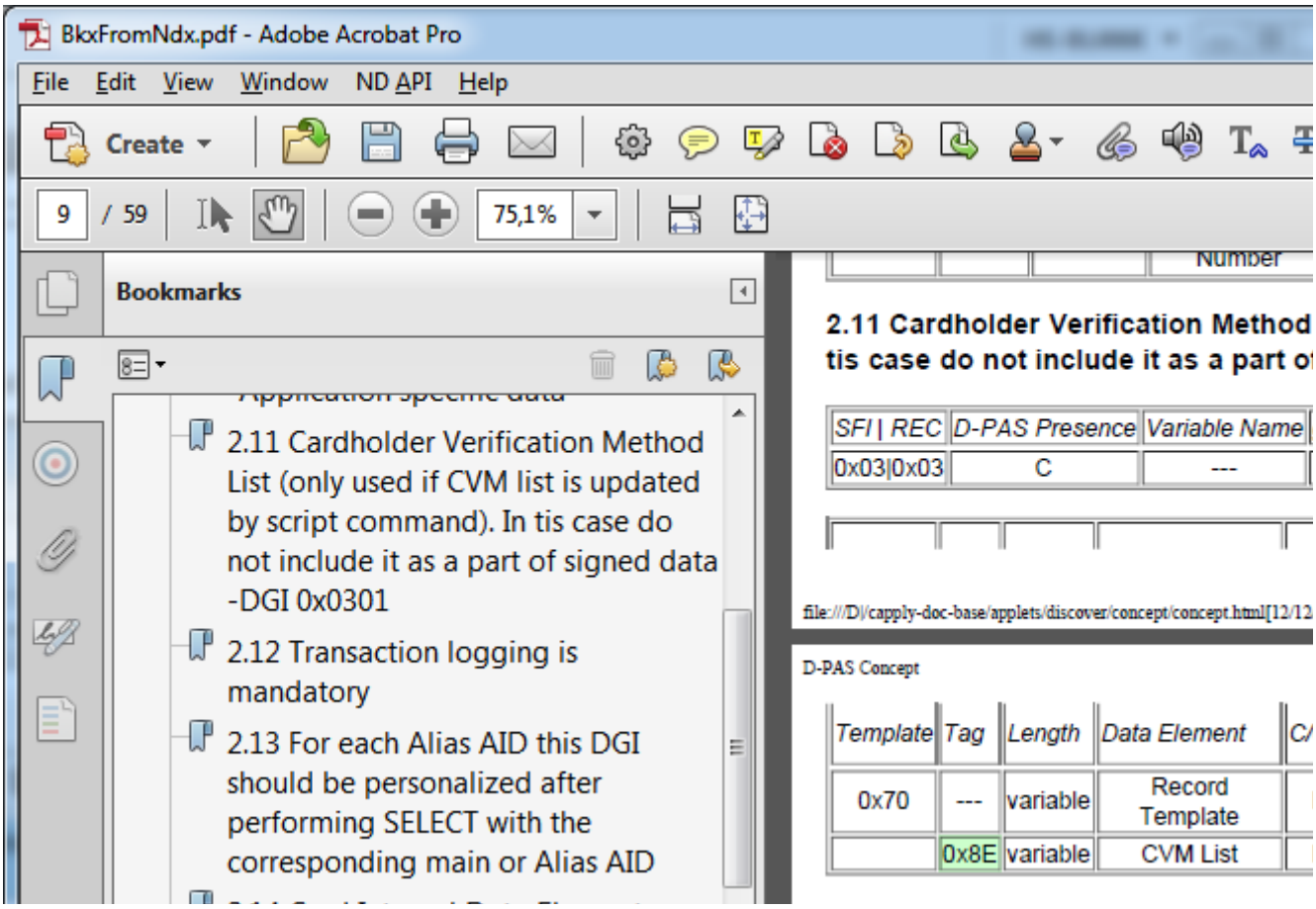
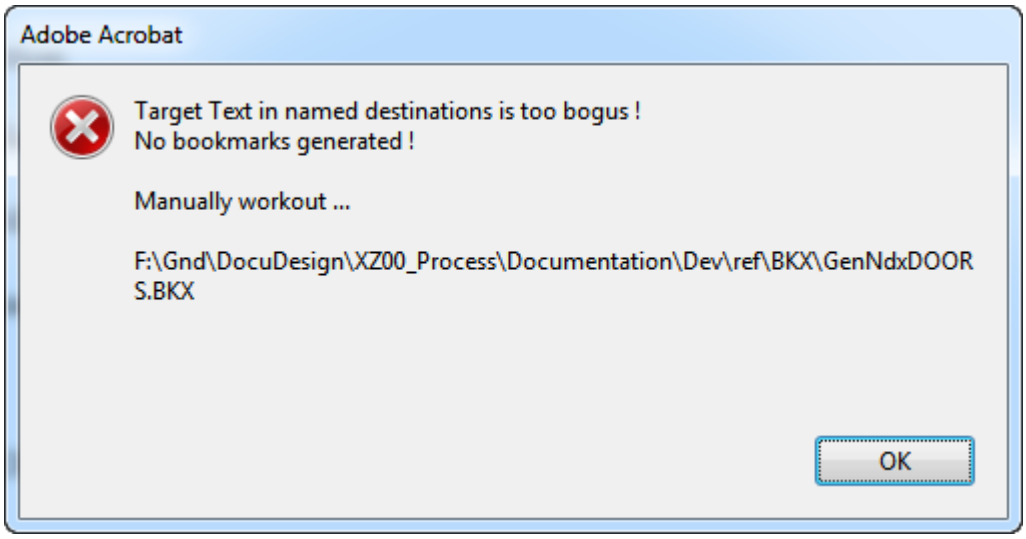


Figure 12-8. Results contains bookmarks as found in the target text

If no or no interpretable text was found in the targets for about 90% of the targets, a warning will indicate the bogus target text and the function is not executed. The warning points to a .BKX (ASCII) file That file can be changed to reasonable content and then it can be imported using 12.1.10 “Export/Import Bookmarks” which also explains the BKX:Bookmark CSV File structure.



Note: After successful creation of the bookmarks you may want to delete the unreadable named destinations and use 12.1.6 “Generate named destinations from bookmarks”.

12.1.8 Generate named destinations/bookmarks from comments

Office 2003 and 2007 PDF conversion allows to “print with comments”. This means, that e.g. a comment in an EXCEL file can become the comment in a printed=PDF converted PDF file.

As it is actually not possible to create *bookmarks* in an EXCEL or a PowerPoint PPT file, there is only one idea how to convert such an EXCEL file into a PDF and still doing bookmarks. This is through comments.

“Generate named destinations/bookmarks from comments” is a special function which allows the generation of *M8.newlink* named destinations from comments. The function is a help for developers which work along a specification. If they want to refer to a place in the specification (PDF) they may place a comment box at the place where they want to set a named destination.

In order to create PDF bookmarks and named destinations from comments, do the following steps

- Write the comment
- Convert the comment

Here we do not give advice how to get comments of a source file (e.g. EXCEL, PPT) into the PDF, we only assume, that a comments is available as pop up with text box.

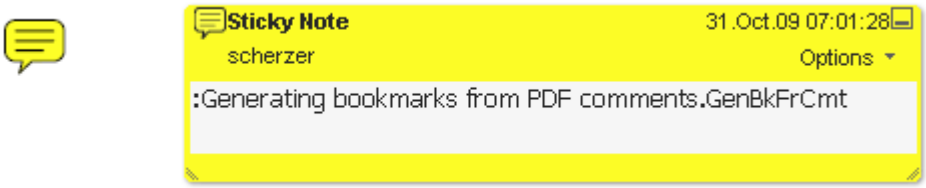


Figure 12-9. Popup and textbox of a PDF comment

The function automatically includes the associated bookmarks at the end of existing bookmarks

The function also generates a **.BKZ** bookmark file which allows to modify the results and import them as bookmarks according to 12.1.10 “Export/Import Bookmarks”.

12.1.8.1 Write the comment

A comment which is subject to be converted is always preceeded by a colon “**:**” as the very first character (no preceeding blanks) of the comment. All other comments remain unprocessed.

Comment Syntax

The full syntax of such a processable comment entry is

```
:[:[:[:]]]<Bookmark text>.<named destination text>
```

Nesting level

The number of colons “**:**” determines the nesting level of a comment. A single colon “**:**” makes a heading 1. Three colons”**:::**” would give a heading three being nested under a heading 2.

Of course the processing order is relevant to determine which `N+1` comment is put under which `N` parent. Processing order aspects are described in the next paragraph.

The conversion does not perform a syntax check on the nesting levels. If you put in a strange order, the results might become as unpopular as your input ordered it to be.

Bookmark Text

The <Bookmark text> may contain any character including colons “**:**” and dots “**.**”.It will be exactly the text that will show up later in a bookmark.

Named Dest

After the <Bookmark text> a dot “**.**” is required to separate the next field which holds the named destination entry. The <named destination text> shall not have blanks or special characters, it will be evaluated and any suspicious character wll be exchanged according to the rules explained in 7.1.4.1 “Step 1: Choose the right destination name” on page 7-88..

Examples: The following examples show some special cases

Table 12-1. Valid examples for convertable comment entries

Entry	Description
`:I become bookmark.WillBeDest`	Normal entry as shown above
`::http://www.test.com.GotoTestCom`	Special chars in bookmark text allowed
`:::Bookmark with dot..BkWithDot`	Dot at the end of the bookmark. Second dot is sacrified as separator to the named destination
`::.OnlyNamedDest`	No bookmark text
`::Only a bookmark` `::Only a bookmark.`	Only bookmark text as there is no dot at all. The last dot gets lost because it is found as separator but since there is not named dest text, only a bookmark is generated
`:Only a bookmark with dot..`	Also only bookmark text with a dot because there is no named destination text

12.1.8.2 Convert the comment

Applying the “Generate named destinations/bookmarks from comments” will visit the comments and create named destinations a the *y-position* of the comment..

The position of the generated bookmark and named destination is taken from the position of the small popup  not from the position of the associated textbox.

y-Position

The order of bookmark aquisition corresponds to the order of appearance. This is not necessarily identical to the vertical position of the comments but can be looked up from the comment panel in the Acrobat Reader. Nevertheless it is the responsibility of the editor to have a proper housekeeping for the correct order.

Auto-comment deletion

Whenever a relevant comment is found, the associated bookmark and/or named destination is created. **Then the comment is deleted**. If you want to preserve the comments, simply save the comments before processing into an FDF comment file.

Preserving comments

After processing, you may delete all comments and restore from the FDF file. The process was designed *with* the deletion as this is more often the desired case and gives a much higher comfort as if an editor would have to delete the special comments manually.

The comment method is fast and has the advantage, that a developer can see the location of named destinations and quickly move them to another point of interest. This technique therefore relates to the requirement Req02: Easy to use.

12.1.8.3 Version changes in target documents

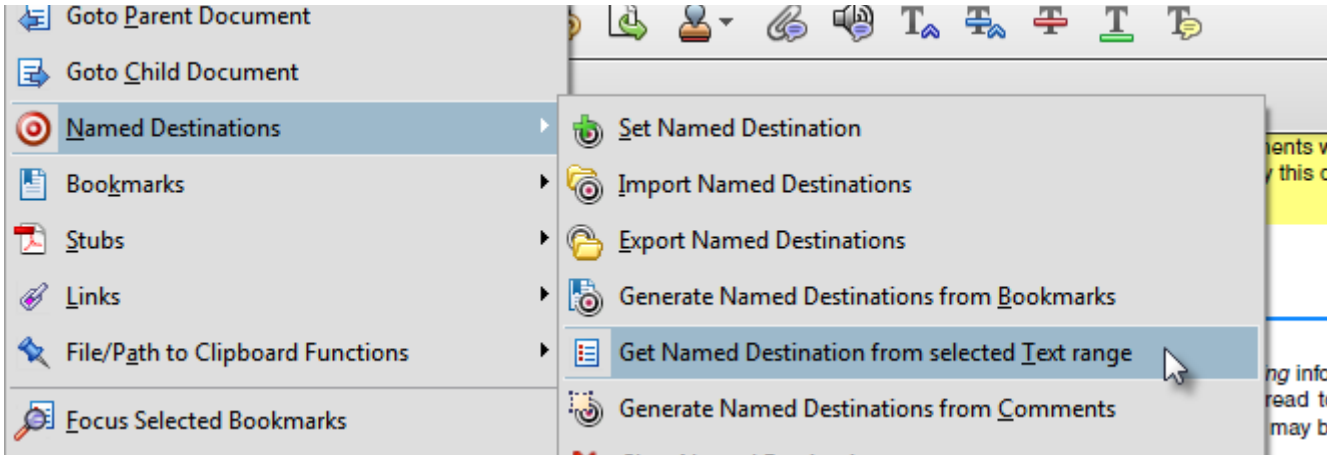
The method to Generate named destinations/bookmarks from comments can also be useful, if the version of the target document is likely to change. There is a fair discussion made in 6.7 “Version management” on page 6-81. In case you foresee changes in a target PDF document and you need more than referencing by chapter, then you may consider the "comment method".

Migrating comments

As Adobe allows the export/import of comments through the FDF files, ("Comments - Export to Data File"), it can be convenient to migrate to a newer PDF version by importing ocmments from the "old" version of a document. Of course a manual rework will have to be done, however, moving comments around might become the preferred choice in several situations.

12.1.9 Get named Destination from selected Text range

The special function section offers additional workflows which address some particular situations. The "Convert DOORS" function allows the automatic generation of named destinations by text/graphic selection.



Text lookup

The generation of named destinations from content selection is described in 7.1.7 “Content based destination generation”.

12.1.10 Export/Import Bookmarks

These functions are helpful in order to create and save bookmarks of a file. A relevant use case is shown in the creation of bookmarks for PDF files that were received without bookmarks (→ 7.1.5.3 “Case 3: No bookmarks in PDF, but Table of Contents” on page 7-94).

"Magnetic" position finder

A special “*magnetic connect*” feature is used when bookmarks are imported. If a bookmark file is imported into the PDF file, a *WordFinder* application scans the target page for the content of the bookmark text. When found, the position of the bookmark is exactly placed to the found text.

This “*magnetic connect*” feature is valuable when importing bookmarks. Sometimes the location of a bookmark is only available by the page number (e.g. if it was extracted from a table of contents using the Toc2Bkx.EXE). To find a more precise y-coordinate the “magnetic feature” positions the actual bookmark after the scan to the correct y-position.

> The magnetic feature can be even used to *improve the accuracy* of bookmarks in a received PDF file. Export - delete - import bookmarks will automatically use the magnetic feature and do the correction.

Sometimes the entries in a table of contents are ALL UPPERCASE whereas the related headings in the text are just normal text. Therefore the comparison on importing bookmarks is *case-insensitive* and even corrects the CAPITALIZED bookmark entries of the *Table of Contents* to their actual appearance in the text.

12.1.10.1 BKX:Bookmark CSV File structure

Exported bookmarks are stored in files with the extension .BKX

First record: The *first line* of *MyDoc.BKX* has a special meaning.

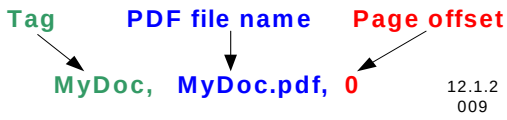


Figure 12-10. Import Bookmarks from Table of Contents

Tag — The **Tag** contains an *identifier* for the document. Although it does not technically hurt, for use with other tools (e.g. Auto-Documentation Tools) this identifier should be unique throughout the entire project.

PDF file name — The **PDF file name** contains the location of the file *relative to the BKX directory* which stores the CSV files (BKX). The BKX directory is created using the "Set/Clear documentation environment" function.

Page offset — The ***Page offset*** field is important here. Some books start numbering with roman letter (i, ii, iii) and continue with numeric restart some pages later (1.2.3...).

Change the ***Page offset*** field in *MyDoc.BKX* to the number of pages *before* the first '1'-numbered page.

Following records: The following records of the BKX file are all of the same structure. Every line indicates a bookmark entry.

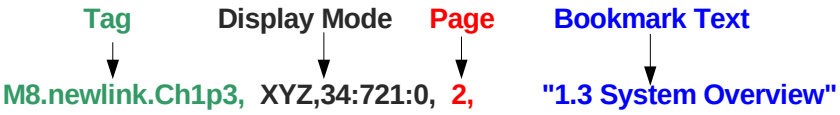


Figure 12-11. .BKX - Bookmarks CSV record

Tag Field — The **Tag** indicates the full name of the named destination used with the bookmark. The prefix *M8.newlink* is required by Adobe FrameMaker. Whenever you insert a hyperlink to a named destination in an Adobe FrameMaker document, then FrameMaker will add the prefix "M8.newlink." ahead of the desired name. Creating a named destination "MyDestination" in a FrameMaker document will actually create a hyperlink to the named destination "M8.newlink.MyDestination".

See 7.1.4.1 "Step 1: Choose the right destination name" on page 7-88 for details for the correct naming of named destinations.

Page Field — The **Page** indicates the page number in the PDF file. This can be different from the page numbering printed on the bottom of the page.

Display Mode Field — The **Display Mode** determines the way the Acrobat reader will display the page when it jumps to the named destination. Typically *FitH* or *XYZ* are the recommended values.

Coordinates — **XYZ** indicates to display the page with the named destination at the top of the page. The view will not be changed from any previous view. In the example above the value 740 determines an offset. The offset is zero-based and the actual number is an Adobe PDF internal number.

FitH — will set the view of the Acrobat reader to *horizontally fit the page* into the left/right margin. As with XYZ, the named destination will be placed at the top of the page.

Other options can be found in the "PDF Open Parameters" documentation.

The **X:Y:Z** component carries the (vertical) y-position of the named destination belonging to the bookmark. The center point is the bottom left point of a PDF page. The actual numbering is an Adobe internal numbering, described in the *Adobe PDF Guide* as part of the freely available *Acrobat Software Development Toolkit*.. (see www.adobe.com/devnet/).

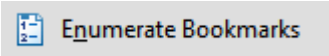
For our purpose we do not *manually* modify the generated y-component. The other components *X*,*Z* are not used by Acrobat, however a special feature is implemented in the ND.API with the Z-component.

Disabling title search feature

The *header search feature* can be switched off *setting the Z-component* of the X:Y:Z triple to a *non-zero value*.

The **Bookmark Text** shows the text as it will apear in the bookmark.

12.1.11 Enumerate bookmarks



Enumerating bookmarks is relevant when a PDF file does not have numbered bookmarks. As we suggest to refer to chapters by their numbers (e.g. *Ch7p3*) numbering bookmarks is essential to allow easy cross referencing. The typical use case is explained in 7.1.5.2 “Case 2: Bookmarks available in PDF, but not enumerated” on page 7-94).

12.1.12 Focus selected Bookmarks

Opening a file often comes with the bookmarks in some unfolded state, showing Head 1 to Head 4 levels. As the reader can become confused, Acrobat offers to collapse all bookmarks to the Head 1 level only. Reading a document this might not be the best choice since also the focus of the currently selected bookmark is lifted.

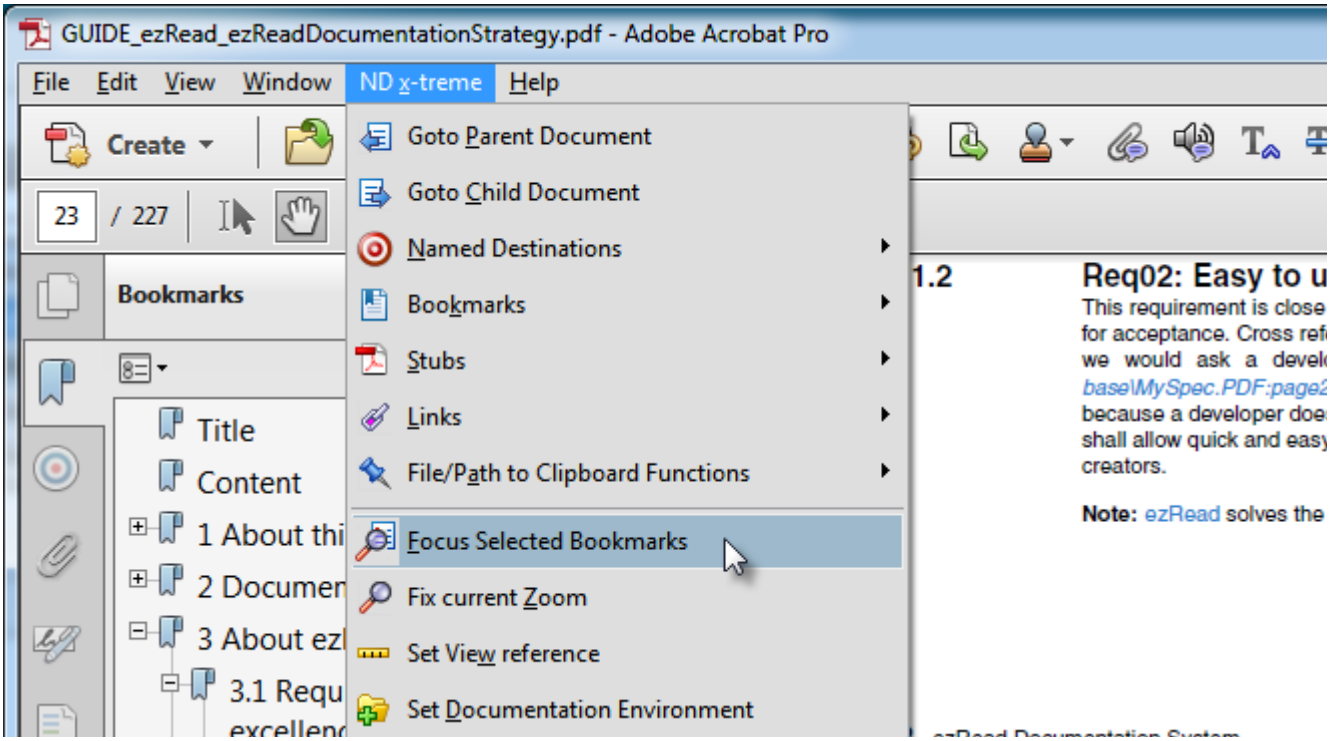


Figure 12-12. Focussing the selected bookmark

The function "Focus Selected Bookmarks" collapses all bookmarks **but the one being selected**. In the above example, chapter 7.1.3 would remain the path to be uncompressed, all other neighbours will be compressed to level Head 1 (here chapter 6).

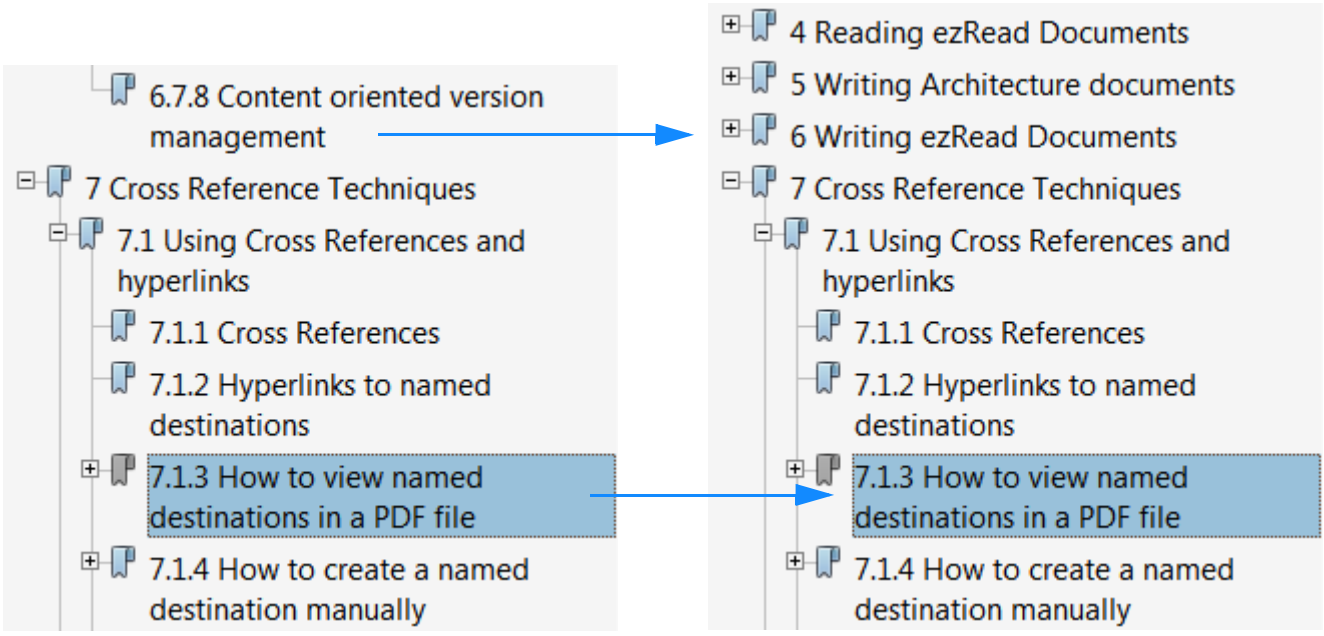


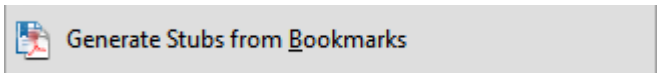
Figure 12-13. Example - Chapter 7.1.3 gets focussed

12.1.13 Generating stubs from Bookmarks

In a SECURE document, it is not possible to place named destinations. Therefore, direct addressing is not possible. The solution is Stub technique described in 7.1.6. The associated use case can be found in 7.1.5.4 “Case 4: SECURED Target document with Bookmarks”.

The function generates stubs in the stubs directory of the topic entered in the diaglog box/

12.1.14 Generating Stubs from named destinations



Documents where named destinations can be applied, do not require stub technique. If, however, references shall be made from documents other than FrameMaker (e.g. EXCEL, MS-WORD, MindManager etc) those programs do typically not support addressing of named destinations. In this case stub technology helps again. However, here the stubs can be used without page/position information.

The ND.API provides a special function “**Generate Stubs from Named Dest**” .

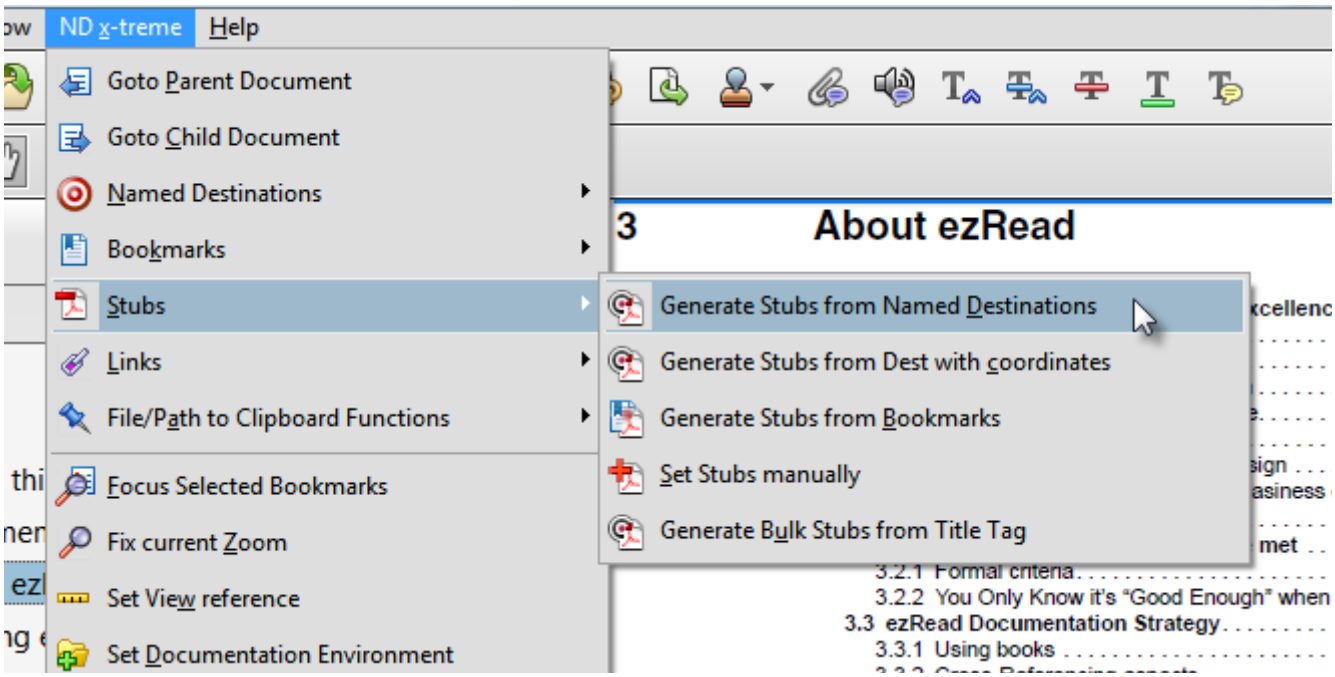


Figure 12-14. Generate stubs from named destinations

This is the recommended usage for documents that do have “*M8.newlink*” named destinations or allow those to be generated by 12.1.6 “Generate named destinations from bookmarks”.

The generated stub files *do not contain page numbers nor position information* since they can resolve simply from the named destination, given in their file name. On opening a stub, the ND.API plug-in recognizes the absence of page/position information and jumps to the named destination, retrieved from the filename.

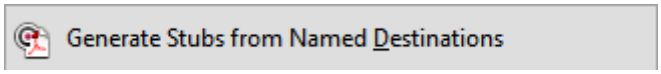
This is also an important alternative when working with Office 2003. Find more under 8.3 “Unwanted artifacts in MS-WORD PDF conversion” on page 8-128.

12.1.15 Generate Stubs from Dest with coordinates

This function does almost the same as Generating Stubs from named destinations with the difference that page and **Y-coordinates are enforced**. This enforcement is typically necessary for SECURED files only as described in 7.1.5.4 “Case 4: SECURED Target document with Bookmarks”. However there might sometimes be a good reason to enforce the hard coordinates, e.g. if named destinations may be subject to deletion later.

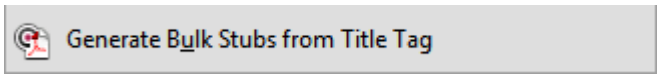
Find some explanation about the page/Y-coordinate mode from Figure 7-22.

12.1.16 Setting Bookmark Stubs manually



In the use case "Case 6: SECURED document - no bookmarks nor TOC" it is necessary to set stubs *manually* since no other information is available to generated stubs automatically. The case is very seldom, however, it may occur. The generated stub contains page/y-position information and therefore does not need to touch the SECURED file. Details on usage are found in the use case description.

12.1.17 Bulk generation of stubs from document title



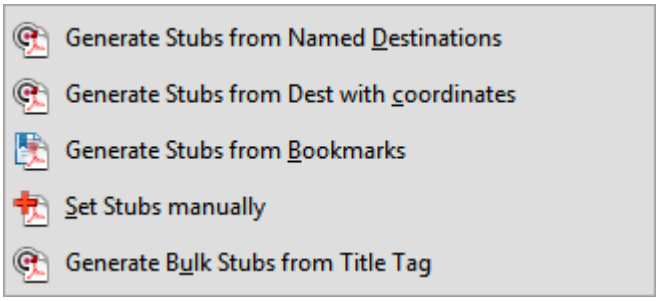
If you named your documents according to the Reference Document File Name Format in chapter 9.2.2, then there will always be a **_Prefix_** separated by two _ underscores.

This can be efficiently used to automatically **create stubs for all documents** that revside in the current document's directory. Or in other words – if you want to create stubs from all documents in a directory, just open any of them in that directory and execute this function.

After that, the stub (STB) directory will contain stubs to all files in the directory, that were named with the **_Prefix_** construction according to 9.2.2.

The actual stub identifier is equal to the **Prefix**.

12.1.18 Stub management



It often occurs, that a document is SECURED. Then it cannot be changed, neither is it posisible to add bookmarks or named destinations. Unfortunately this often happens on the most important external documentation like standards or data sheets.

A document which has no named destinations *can normally not be referred to by chapter numbers*. As you cannot add named destinations to a SECURED document (neither generated them from bookmarks) this seems to be a problem.

Since April 2009, a new 'stub' technology has been added to "The ND.API plug-in". This allows a straight addressing of SECURED documents by its chapter number although it is not possible to write any destination in the protected target document. Uses are described in

- 7.1.5.4 "Case 4: SECURED Target document with Bookmarks" on page 7-97
- 7.1.5.5 "Case 5: SECURED document with TOC but no Bookmarks" on page 7-98
- 7.1.5.6 "Case 6: SECURED document - no bookmarks nor TOC" on page 7-99

12.1.19 Create Hyperlink from selected text

Creating a hyperlink to a named destination of another (or the same) PDF document with with the usual methods of Acrobat is a very annoying process. The ND.API plug-inhas significantly the improved this process which now can be done in a few seconds compared to about a minute with the Adobe method.

> The classical Acrobat method would require to select the "Link" tool in the "Tools" section (Acrobat X), then mark a word or region. From there a panel opens and "Go to a page view" needs to be selected. Annother prompt opens and will wait until the user opens the target document, places the correct target and presses "Set" in the prompt to finally set the named destination and the link. Try it – and get frustrated enough to find motivation to read on for the solution provided by the ND.API.

Using Stub management a document and its chapter can be easily specified by a small text, opening the target document and alignment of the target are no more necessary. The ND.API plug-in allows the creation of links by two methods

- Create Hyperlink by text selection
- Create Hyperlink by region selection

12.1.19.1 Create Hyperlink by text selection

Text selection is a common feature in all versions of Acrobat. You may press the  button which allows to select text the common way. The following example will explain purpose and technique to create these hyperlinks efficiently.

Example "Comments"

In our example the editor of a CEN standard has to integrate the French comment #12 ("replace **SOF analysis** by **analysis**") into the working draft of the standard. While the editor does so, whenever he has accomplished one of the comments, he places a named destination ("Bookmark" in MS-WORD) in the working draft of the standard "*Part1.docx*". In MS-WORD the editor uses the "Insert Bookmark" function as described in 8.1.5 “Using MS-WORD bookmarks”. Our editor is very practical and (s)he names the bookmark "FR12"

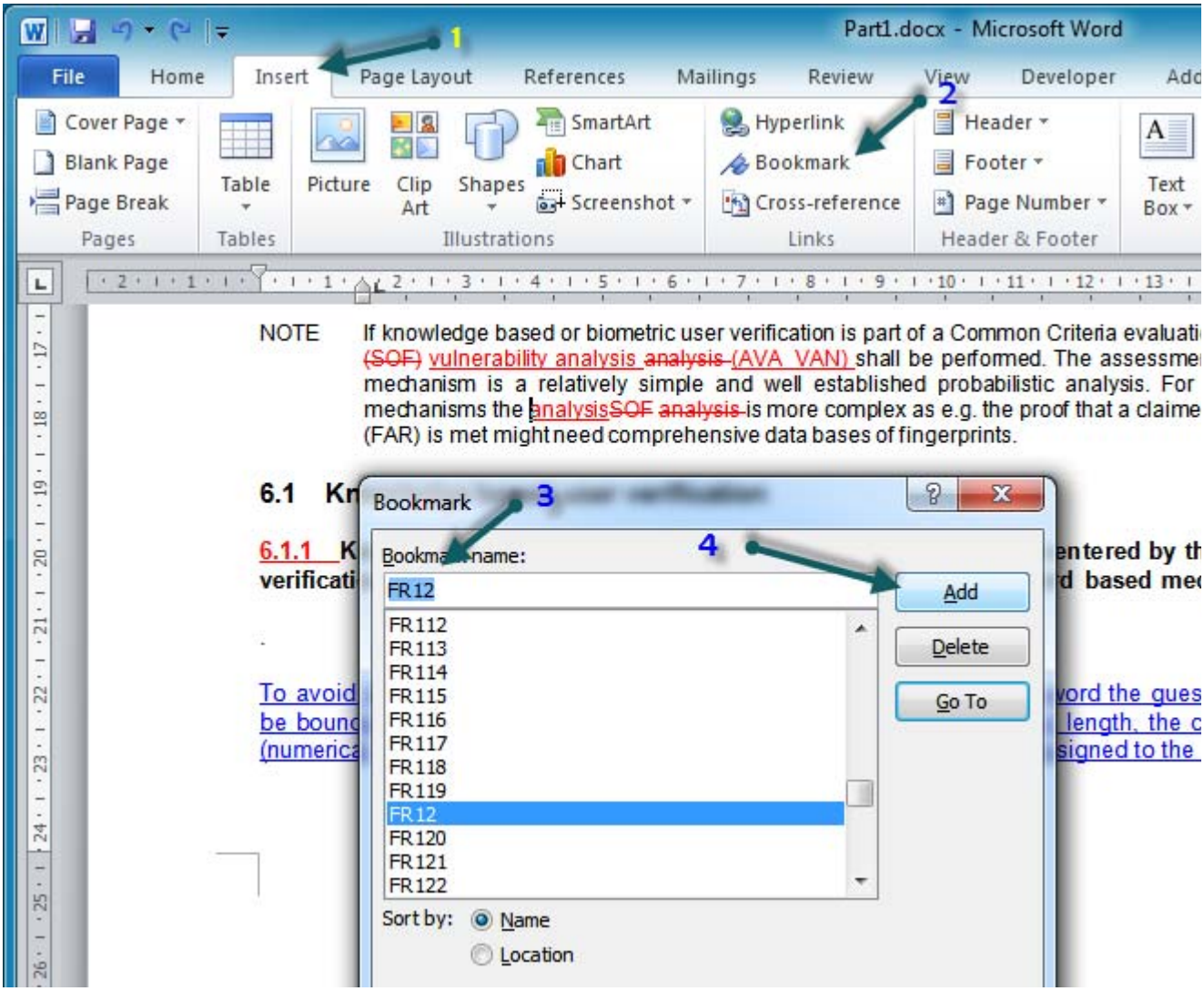


Figure 12-15. Convert Links Choices

Now there will be a named destination "FR12" in the final draft of the standard. Our editor may now save "Part1.docx" after (s)he has created a version "Part1.PDF". The editor has to consider the artifacts that MS-WORD adds to a PDF document (→ 8.3 “Unwanted artifacts in MS-WORD PDF conversion”) and repair them with the function described in 8.4 “Converting MS-WORD links”.

Then (s)he applies the function “Generate named destinations from bookmarks” to make named destinations, followed by stub generation using “Generating Stubs from named destinations”. Our editor uses "**P1**" as document locator, hence there will be a many stubs available, one of them named **P1_FR12**. All the above steps have to be done once only – of course our editor first works in all 142 French comments (with their bookmarks) and produces the PDF after all that work.

Creating links in the comment paper

The comment paper from the French delegation had been delivered as PDF. Now the editor wants to comfort the French delegation allowing them to find the changes of the Part1.PDF in their comments. Of course the French delegation did never add a link in their comments, neither did they know what to link because before the editor did the change, there was no target to address.

So the editor wants to add a hyperlink to the comment paper. He will open the comment paper and mark the French comment #12.

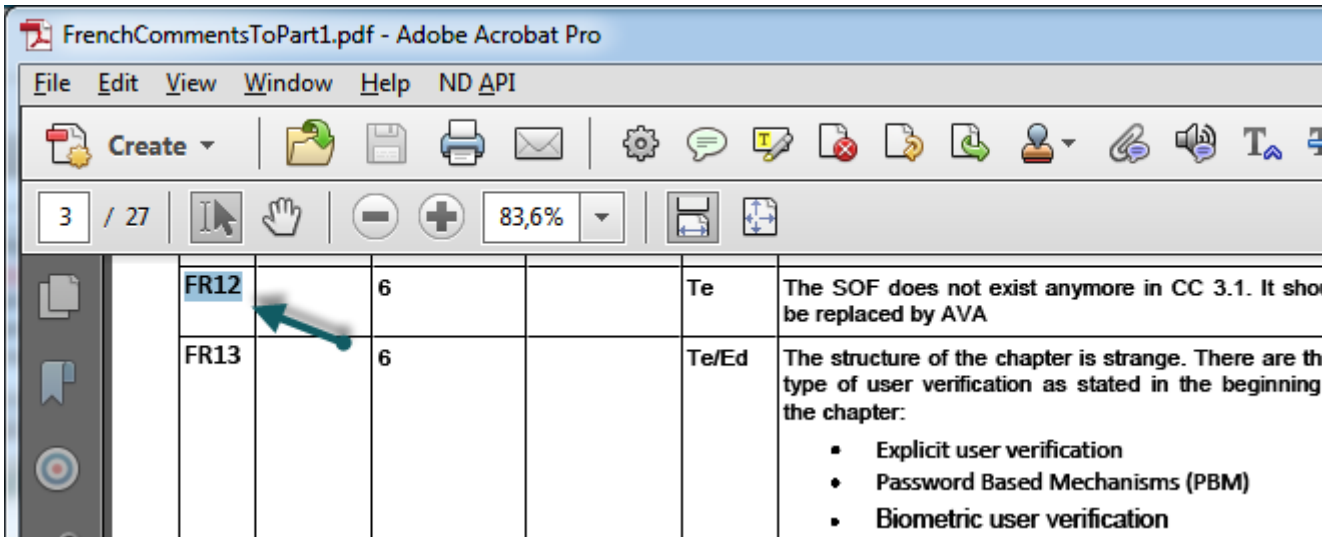


Figure 12-16. Marking the text of the French comment

Now the editor uses the function of the ND.API 

A dialog box opens ...

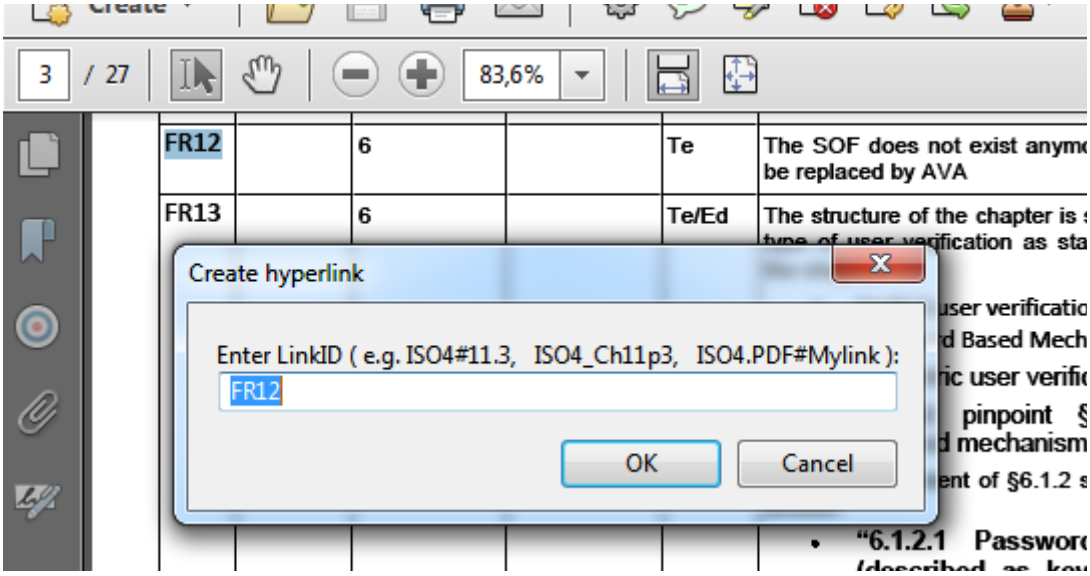


Figure 12-17. Hyperlink dialog box

The box suggests several methods to address a stub, our editor enters ...

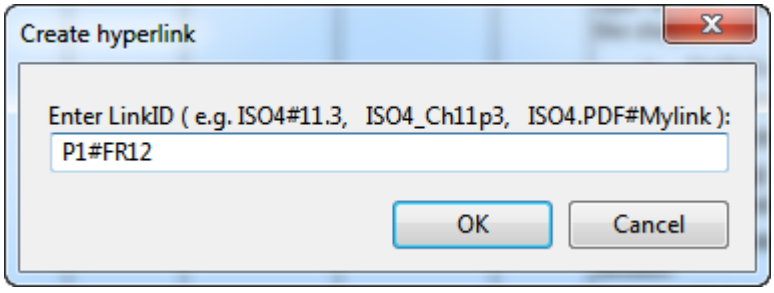


Figure 12-18. Editor's choice to hyperlink

... and the ND.API will generate a link to the (existing) stub **P1_ChFR12.stb**.

Automated link generation

If the comment paper has the comment entries always on in the same position, the editor might even add the links automatically using the function described in SHIT12.1.9 “Get named Destination from selected Text range”

Test the link

The editor can immediately test the link selecting the "FR12" text. As a matter of fact, the process is very fast. Having the "P1#" in the clipboard, the editor can quickly create hyperlinks in seconds. A real use case required approx. 11 minutes to add 480 (!) links into a comment paper.

Send the package

Now the Editor can send the entire package – new Part 1 and the French comment paper back to the French delegation and they can easily find the changes by just clicking on their own comment. To build the delivery package for the French delegation, the editor uses the “Exporting linked files” function described in chapter 12.1.26.

The above example shows a typical workflow to create hyperlinks. Intentionally the function links to stubs as Stub management allows an efficient adressing. With the export function (“Exporting linked files”) those stubbed links will be converted to regular links, so there is no disadvantage to maintain stubs at the editor’s level.

> Of course, links into PDF files are only advisable, if the original source does not exist. In any other case, where the source is available as DOC(X) or .FM, then there is no point to add the hyperlinks on PDF level – well, except for the idea that generation of links is much faster in the PDF than it is manually in the original source DOC(X).

12.1.19.2 Create Hyperlink by region selection

With Acrobat X and later it became possible not only to select text but also regions. The same setting using  allows to mark a region if the editor starts on a non-character location.

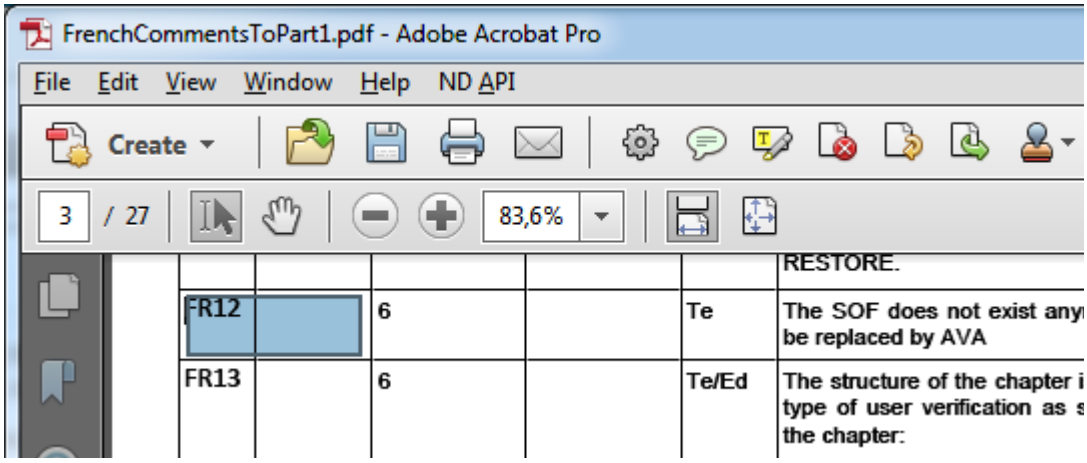
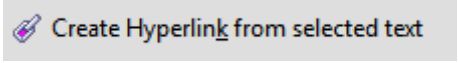


Figure 12-19. Editor selects a region

Using the 

function will open the same box as in Figure 12-18, but with no suggested entry. The editor may enter the same text as before (e.g. "**P1#FR12**") and a link on the entire region will be added. This can make hyperlink selection even faster because you have larger fields (easier to hit with the mouse) but sometimes it is a bit annoying to select regions because the Acrobat tries to make a text selection, if possible.

12.1.20 Convert links

MS-WORD allows to link to external files through the “Hyperlink” function. Usage of this function is eplained in Linking files with MS-WORD. On this way, several artifacts in the final destination are produced. While Office 2003 had quite a number of nasty effects, Office 2007 has improved significantly. All present Office 20xx versions do still have the artifact that MS-WORD creates URLs as links as described in 8.3.2.

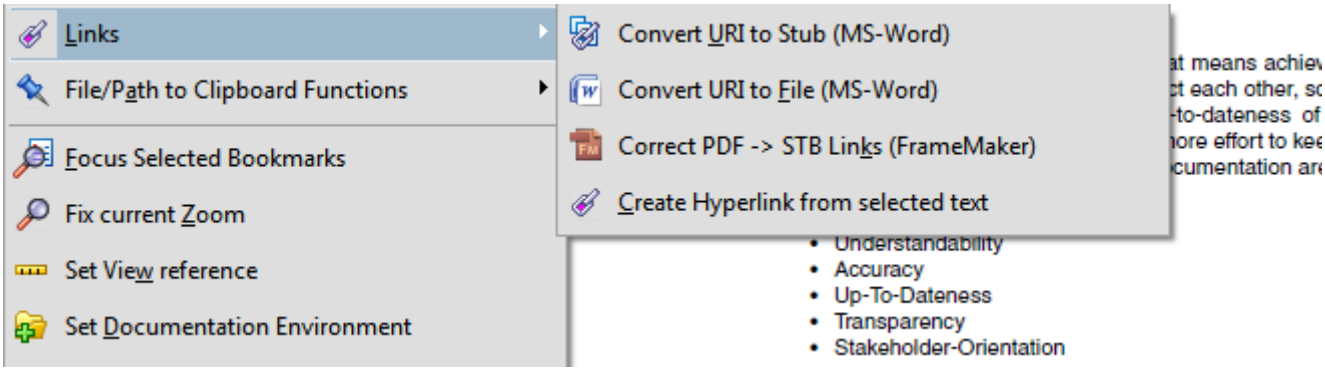


Figure 12-20. Convert Links Choices

The following post-production features will correct all the artifacts described above.

12.1.20.1 DITA post processing

One of the great things we can do is to create links in a VISIO graphic, export as SVG and the links will be still contained in the final PDF when used in a DITA document.

However, these links do not actually know their target because during processing the DITA-OT changes the names of the target with a prefix (e.g. *unique_7_connect_42_<name>* in order to avoid ambigous targets.

The function ND-xtreme → Links → DITA post processing will repair these links and therefore shall be applied on the final PDF. It implicitly converts URLs to direct path links.

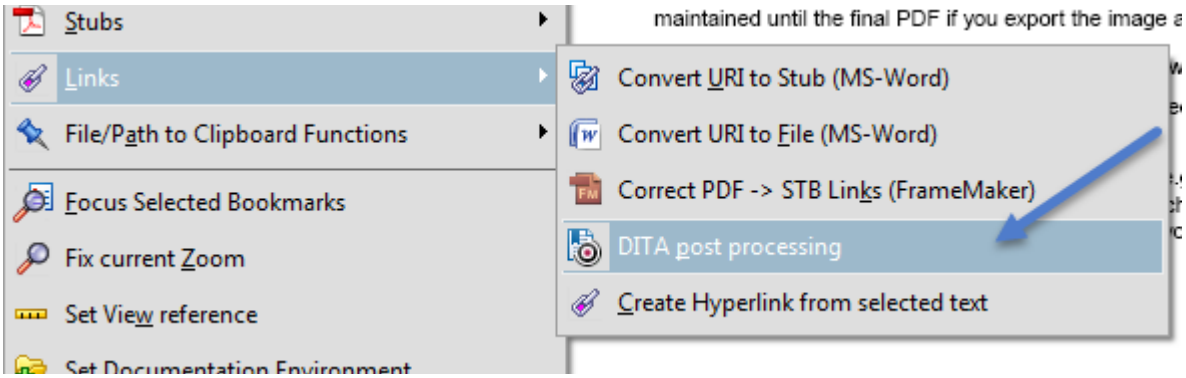
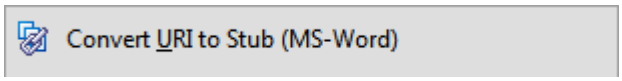


Figure 12-21. Convert Links Choices

DITA named destinations

Any id-attribute given in DITA that starts with an underscore _ will be converted into an M8.newlink.<name> named destination. The underscore will not be part of the final named destination, it is used to make the ND.API plugin recognize that this destination shall be converted.

12.1.20.2 Convert URI to stub



MS-WORD's URI (indeed it is called URI not URL) links will be converted to regular PDF links. If the link went to a stub file (→ Stub management) then a regular PDF link to a stub file will be maintained in contrast to the the Convert URI to PDF–function which will interpret the stub file and establish a direct link into the stub's target file.

This function is recommended in most cases, where further modifications are to be done, in particular, if the target file will be modified and its stubs are likely to be (re-)created later than the linking file (that which contains the links) is produced. The final conversion from stubs into PDF links is automatically done with the "Exporting linked files" function.

12.1.20.3 Convert URI to PDF

MS-WORD's URI links will be converted to regular PDF links. If the link went to a stub file (→ Stub management) then a regular PDF link to the stub's associated PDF file will be created by interpretion of the stub – in contrast to the the Convert URI to stub, which maintains the link to a stub.

If the stub points to a SECURED file, this function works exactly like the Convert URI to stub function because the target file is not expected to have useable named destinations.

12.1.20.4 Technical background on MS-WORD link conversion

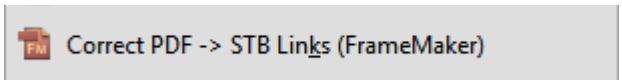
The following description gives a brief detail about the filter functions

- Linking files with MS-WORD will be converted from URL to a *PDF link*.
- A Link to external MS-WORD document will be converted from URL to *PDF link* changing the extension of the target document from ".DOC" to ".PDF"
- A Link to external PDF with named destination will be converted from URL to *PDF link with named destination*. The "M8.newlink"-prefix will be added to the link unless it is already part of the link.
- A Link to external MS-WORD document with DOC bookmark will be converted from URL to *PDF link with named destination* changing the extension of the target document from ".DOC" to ".PDF". The "M8.newlink"-prefix will be added to the link unless it is already part of the link.
- *Internal Links* will be reconfigured from "*Goto XYZ*" links to stabil *PDF links* to named destinations. The appropriate named destinations will automatically be added to the file.

  The "M8.newlink"-prefix will be added to the link to allow the created named destination (internal target) also be addressable from the external world. This avoids the artifact described in 8.3.3 "MS-Word produces unstable links" on page 8-130.

The actual change mechanism in the Adobe PDF file is complicated and shall not be explained here. However it is recommended to use the File-SaveAs function after conversion in order to reorganize and consolidate the changes.

12.1.20.5 Convert PDF to STB (FrameMaker)



Target documents to be referenced may be *SECURED*. This means, it is not possible to add named destinations so those documents. The ND.API provides a special solution to this problem described in 7.1.11 “Linking FrameMaker to SECURED documents” on page 7-108, providing Stub management.

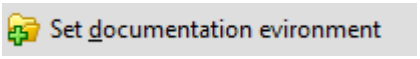
When using Stub management, links created to stubs will be converted by AdobeFramer into links to *PDF files* as a general rule of PDF production (printing). While this is a valuable feature if links to other FrameMaker files are printed (and converted to links to PDF), this mechanism is explicitly not desired with links to Stub files. Therefore the final (FrameMaker printed-) PDF file needs to be corrected to point to stubs again.

The function Convert PDF to STB (FrameMaker) performs the appropriate corrections to the FrameMaker generated PDF Files. Find more about the details of those artefacts in 7.1.11 “Linking FrameMaker to SECURED documents” on page 7-108.

After the conversion, a link (e.g. DCG_Ch5p2.***pdf*** – *wrong!*) will point again back to its original destination (e.g. DCG_Ch5p2.***stb*** – *correct !*).

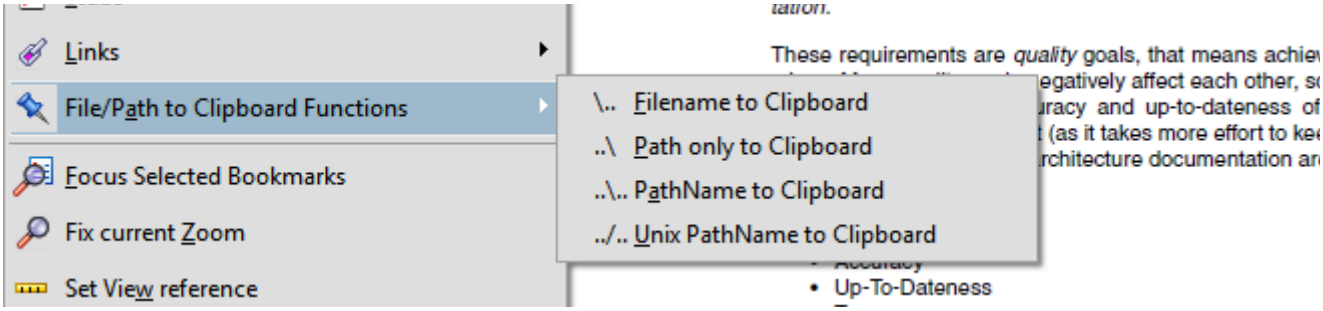
> The mechanism does work for other extensions as well. If you have linked an EXCEL file (e.g. Stat.XLSX), FrameMaker printing would actually change the link to search Stat.PDF.
>
> The Convert PDF to STB (FrameMaker) searches all files of a target directory to match the file name (here "Stat.*) and replaces the extension of the link by a uniquely identified file. If in our example "Stat.XLSX" and "Stat.XLS" did exist in the target directory, then the function will not change the link since it cannot identify the correct target.

12.1.21 Set/Clear Documentation Evironment



This function sets the documentation environment. Details of this philsophy are described in 11.7 “Set/Clear documentation environment” on page 11-178.

12.1.22 File/Path Function



These small helper functions simply copy the present documents filename (including the extension) into the clipboard. This may come in handy sometimes when links shall be generated in documents.

Filename to Clipboard: copies the **filename only** – no path information

Path only to Clipboard: copies the **only the path** in Windows notation (backslash) "...\...\..." – suitable when the documents directory shall be selected later.

PathName to Clipboard: copies the file with the **entire path** in Windows notation (backslash) "...\...\..."

Unix PathName to Clipboard: copies the file with the **entire path in UNIX notation** (slash) ".../.../.../", which is also the prefered method in Adobe FrameMaker notation. The colon of the drive letter, however, will not be removed (we are still in Windows ...).

12.1.23 Set ND view reference

Normally Adobe Acrobat aligns the named destination to the top of the window. For the readers convenience this can be changed with the function

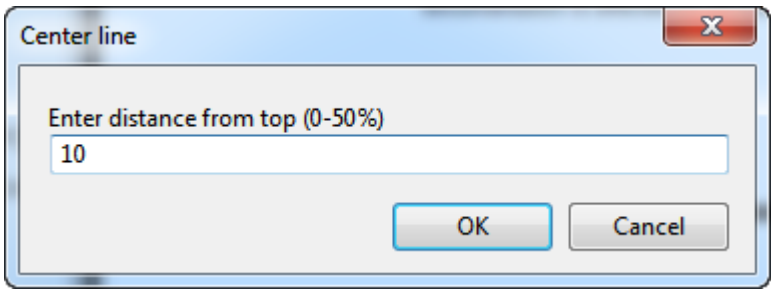


Figure 12-22. Position of a named destinatvion with 10 % offset from top

A prompt will ask for the amount (in %). A value from 0 to 50 % is allowed, higher values do not make sense because a reader is typically interested in reading things *below* a named destination.

Setting the value “0“ switches the feature off

With the view reference set, a linked document’s named destination will no more be placed at the top of the window but with the *selected offset*. In order to allow an intuitive recognition of the destination a *blue arrow* will be shown at the left margin to indicate the exact position of the destination.

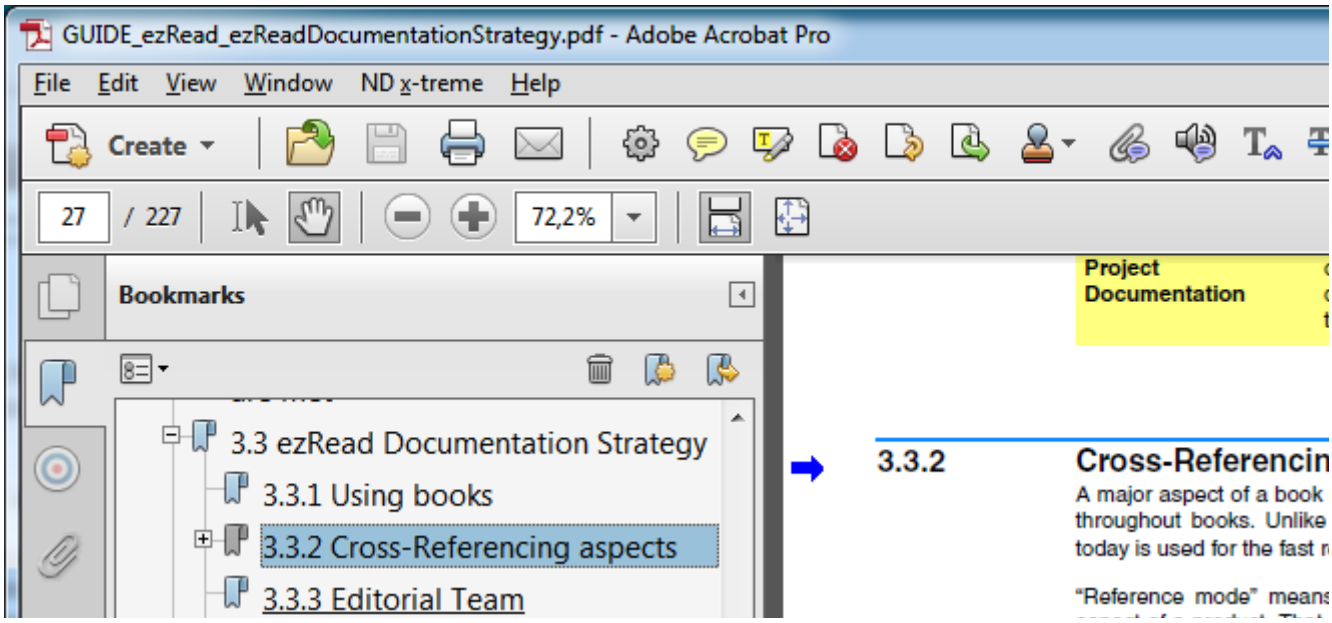
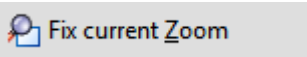


Figure 12-23. Position of a named destination with 10 % offset from top

This marker is not part of the document and will disappear with the next drawing cycle of the window content.

> The view offset will be stored in the registry under *HKEY_CURRENT_USER\NDplugin* and is therefore available as a permanent user setting. (Can even be changed there).

12.1.24 Fix current Zoom

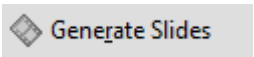


This function reads the present zoom factor and sets it as a fixed value.

Fixing the zoom factor to the present value is halpful if the zoom is set to *Fit Width* and the PDF *document has some pages in **landscape** format*. Then the *Fit Width* display switches to show the entire landscape frame in the present Window which often results in too-small characters. Then, even when viewing a *Portrait* like page layout the *Landscape-Fit* remains active since Acrobat remembers the presence of landscape in the current document.

To avoid switching, the *Fix curent Zoom* can be used, Regardless from what size of page is encountered in a document - the zoom value is no more changed automatically.

12.1.25 Generate Slides



This is a handy function to create bookmarks on PDF printouts of PowerPoint slides. A set of Bookmarks, enumerating from 001 ... 999 will be placed exactly on the top of the slides.

This can be helpful to address a particular slide of a presentation from another source – as Stub management can be used, any slide of a full presentation may be easily accessed by the help of this feature.

12.1.26 Exporting linked files

In order to export a tree of files (typically starting from a host PDF file), the ND.API provides a powerful export mechanism. Files will be exported from anywhere they are linked - to two destination directories (BOOKS and REFERENCES). The links within the files will be changed accordingly. See 7.2.1 “Using the ND.API Export function” for details on using this export function.

12.1.27 Clearing export directory

The export directory can be cleared of any files and subdirectories to allow a fresh export. Find more about the usage in 7.2.5 “Clearing the export directory” on page 7-116.

12.1.28 Remove Personalization

This function removes unwanted personlization information and watermarks. Of course the document shall not be SECURED.

Watermarks can be very nasty because they distract from reading and spoil the intuitivity of the actual document.

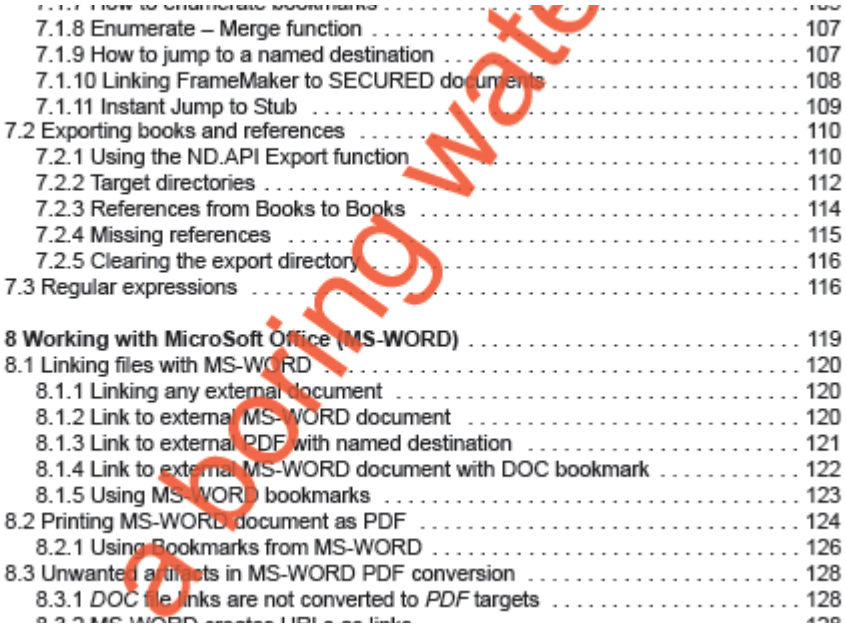


Figure 12-24. Example of a Watermark spoiling readability

The Remove Personalization removes watermarks by feature recognition and object deletion.

Typically two types of watermarks are applied. The easier one is a text object which is overlayed on every page. Deleting the text object on every page removes the Watermark.

Another implementation places several thousand graphic elements in order to form the actual background watermark.

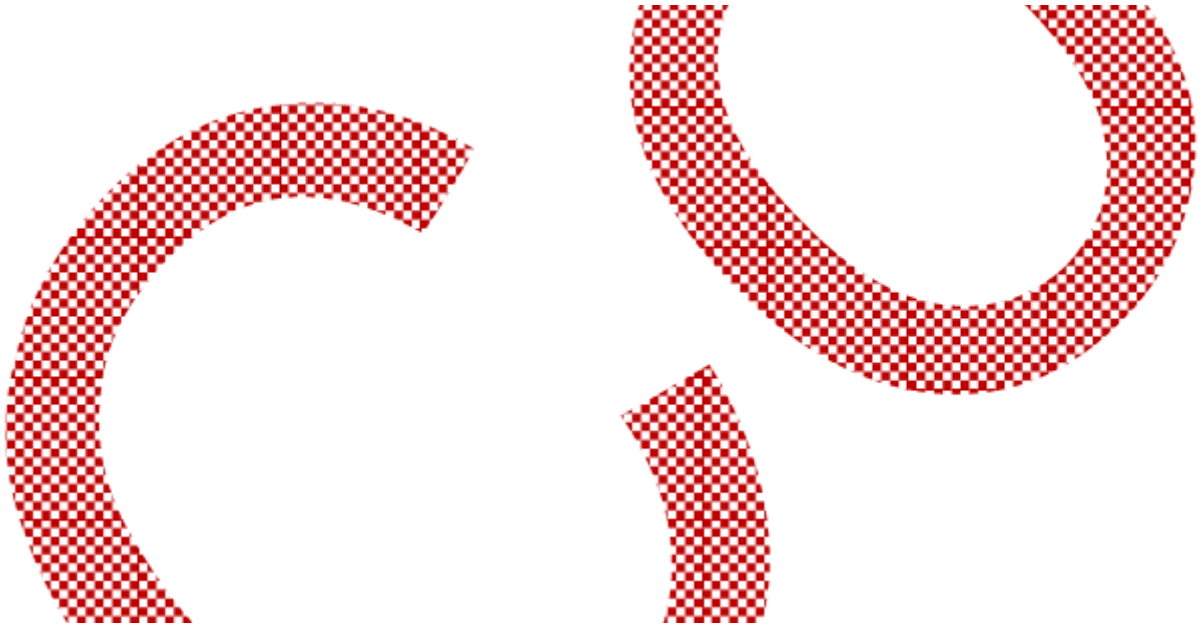


Figure 12-25. "Confidential" watermark by graphic elements

The Remove Personalization function may remove both types, yet the user interaction needs to be slightly different.

In order to remove a watermark, the following steps need to be taken.

Select application range

The text or graphic box within the removal shall be done shall be selected.

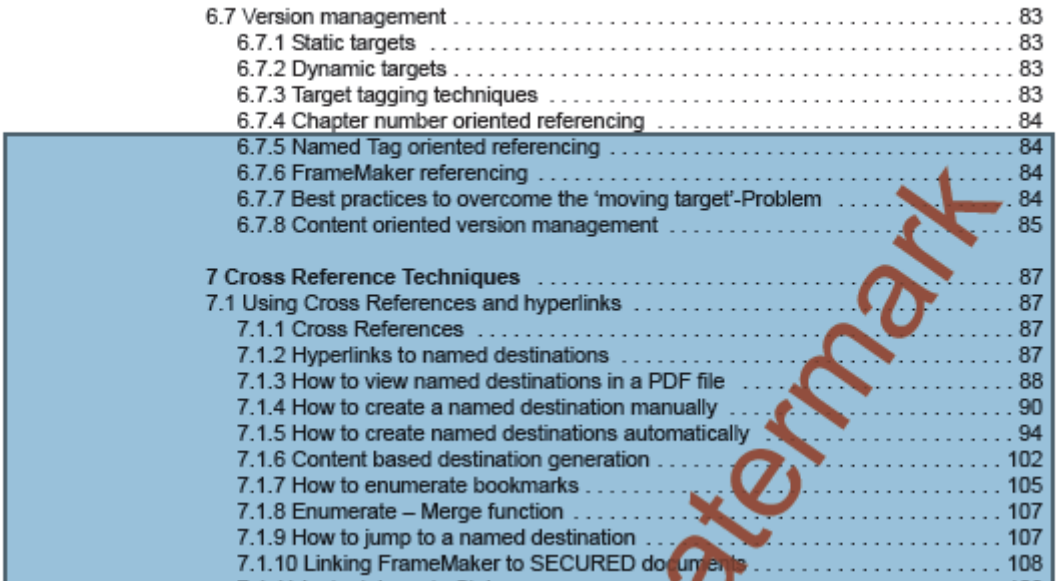


Figure 12-26. Select application box

Figure 12-26 shows the selection of a range by a graphic box. This selection format is only available from Acrobat X. However, it is also possible to select a text range.

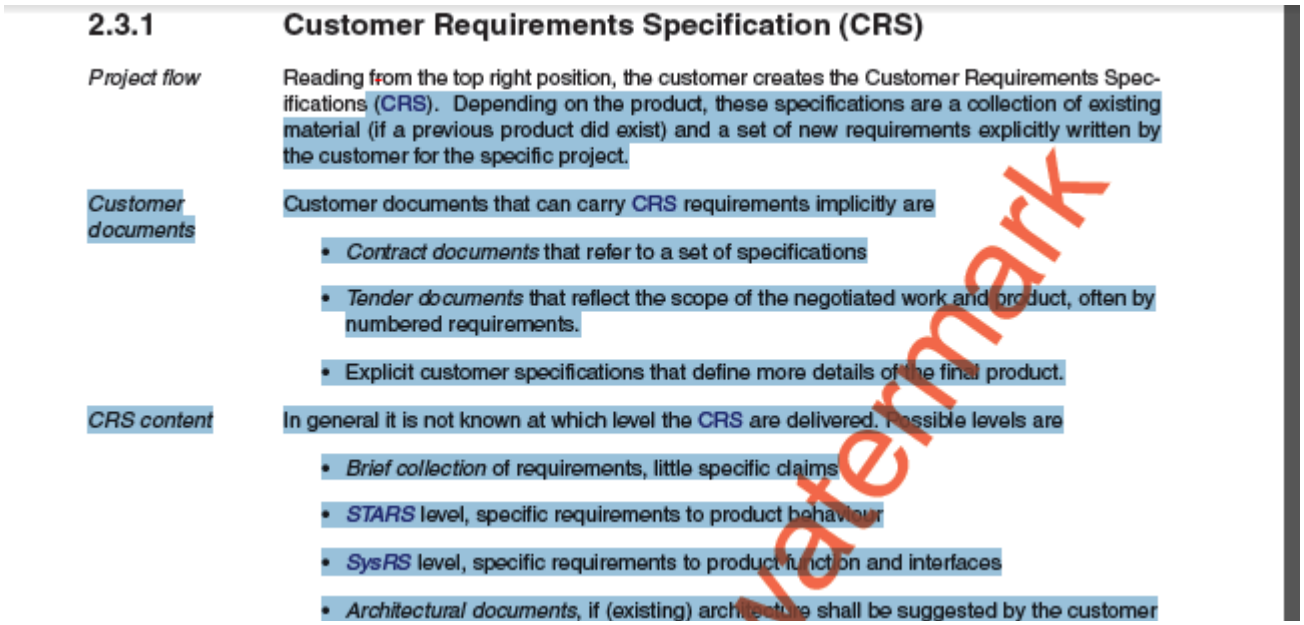


Figure 12-27. Text selection

Text selection shall be made on a page that contains enough text to span over a range that includes the watermark. Select any page with much text in order to find such a situation

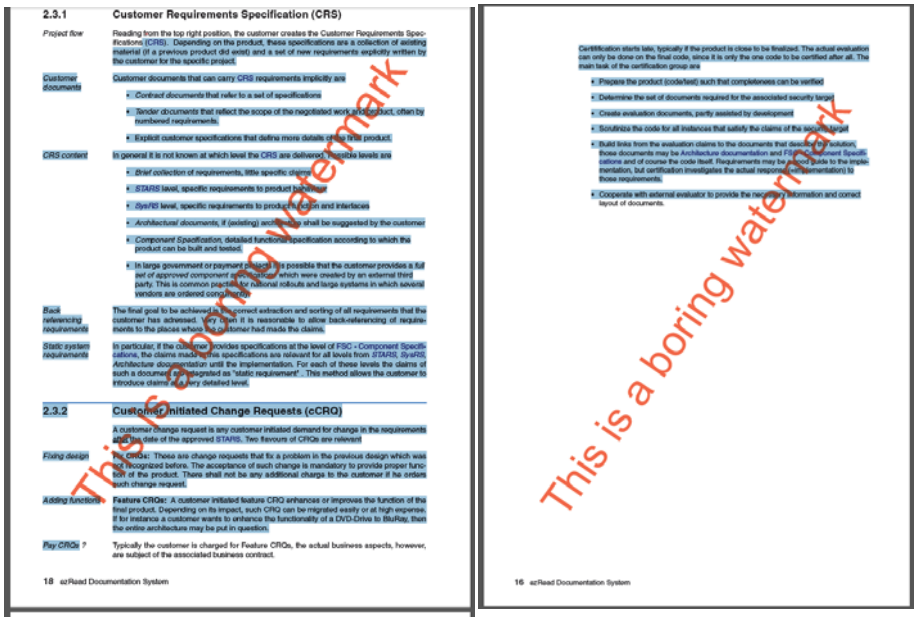


Figure 12-28. Sufficient (left) text selection compared to insufficient text selection (right)

Figure 12-28 shows the difference between a page that allows selection coverage through text compared to a page where there is not enough text to make a proper selection.

Launch function Select the "Remove Personalization" function

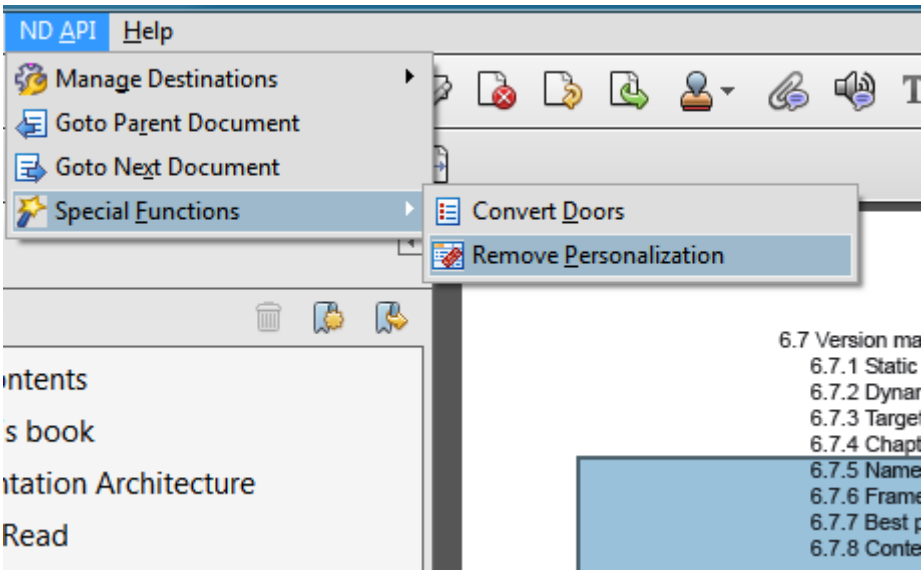


Figure 12-29. Select function

A dialog box appears to let you specify the page range an text pattern.

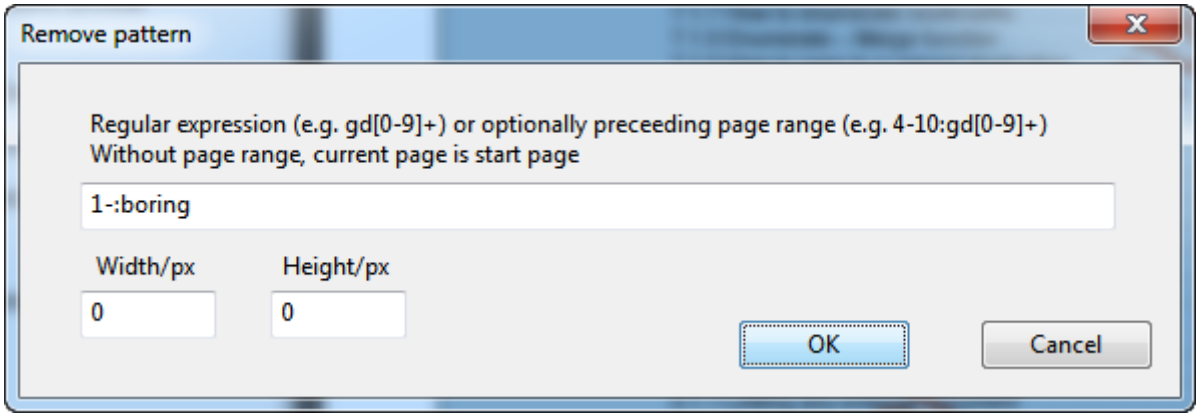


Figure 12-30. Text selector

The above example selects all pages starting from the first page ("1–") and removes any object within the range that contains the word "boring". Actually a regular expression can be given which is more powerful as a simple word choosen in the example. The build scheme of regular expression is explained in 7.3 “Regular expressions”.

> The function will also work on any text in the document. Hence do not use trivial words which would affect tha actual content.

For a graphic watermark you need to determine the typical size footprint of the graphic objects. Opening the object view in the Acrobat allows to identify the pattern by which the bookmark is created.

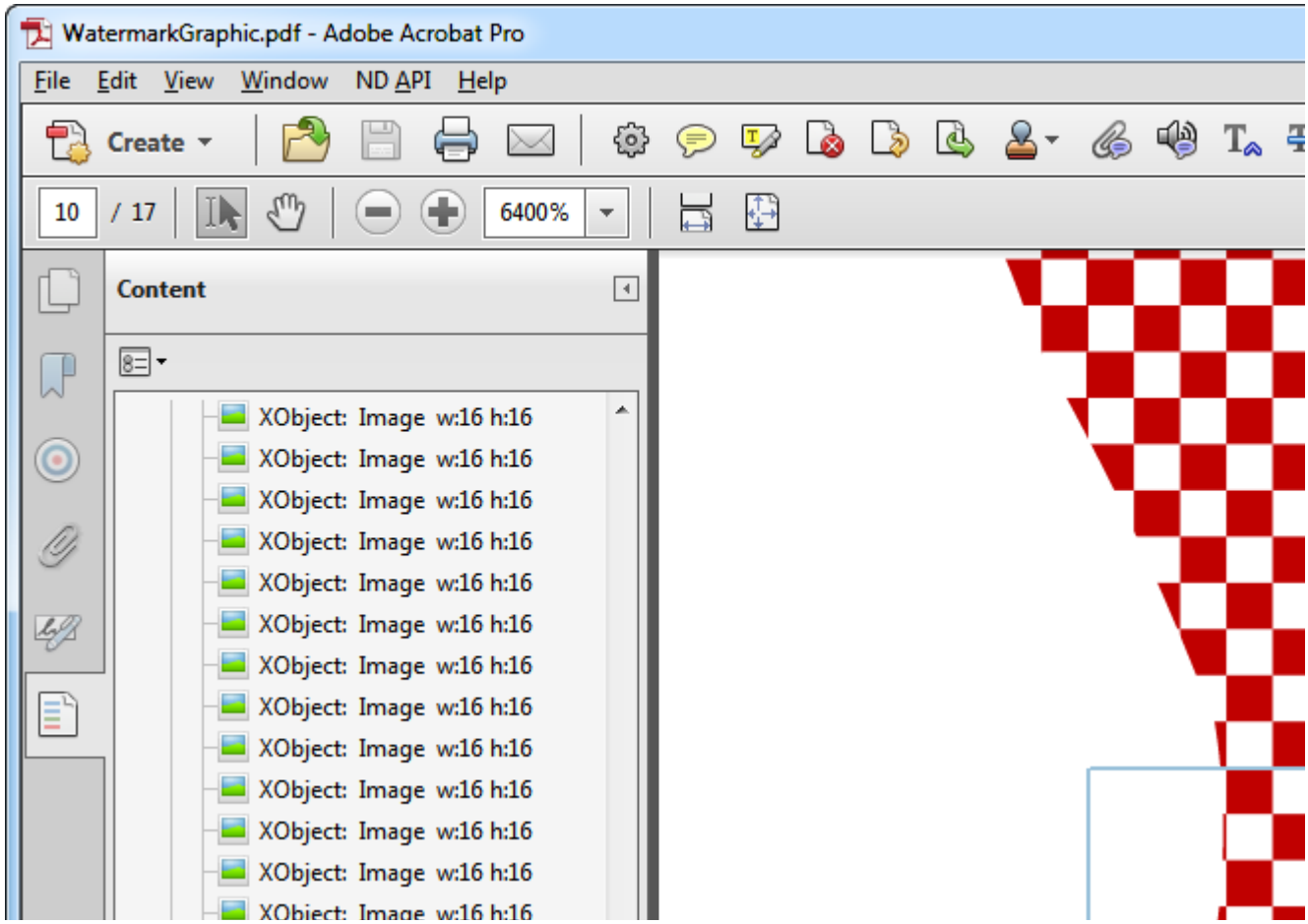


Figure 12-31. Watermark object analysis

The above example shows boxes of size 16x16. This can be used a filter criteria in the dialog box

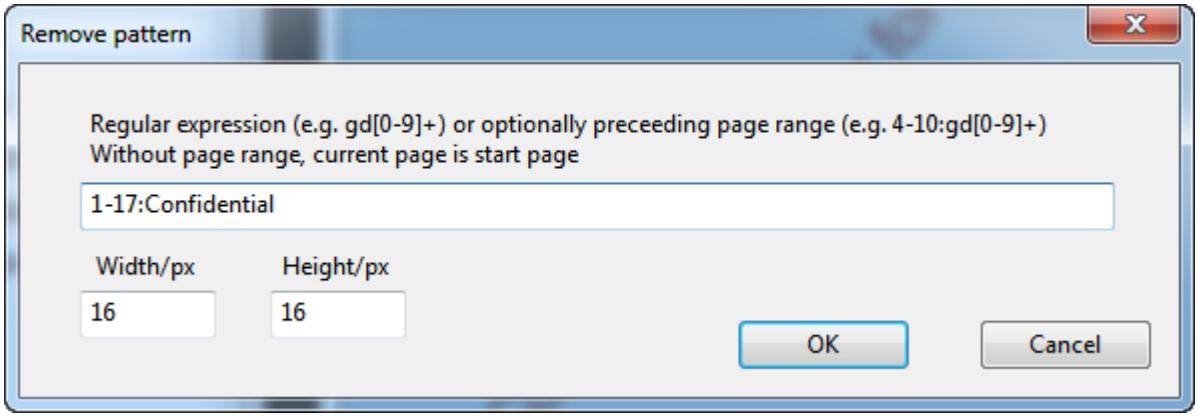


Figure 12-32. Graphic object filter

If the width/height information are non-zero values they will take precedence of the regular expression for the text. However,the page range will be interpreted.

Execute removal Press "OK" to remove the watermark. If you did not start on the first page (because selection was not optimal there) note that the function starts removal on the current page unless the page range is specified in the dialog.

12.2 Accompanying tools

Where to find them

The accompanying tools are installed in the %ProgramData%\ezRead\Tools directory when Installing with 01_InstNd.EXE.

The tools are command line tools which allows them to be used in Batch files. To open a command window use the Windows Start button and enter "cmd" in the search field.

PATH variable

Installing with 01_InstNd.EXE automatically sets the user PATH= variable to the tools, however if the user defintion addes to the end of the HKEY_LOCAL_MACHINE defined path, then the entire path may easily extend 256 bytes and will no more be recognized by the command prompt. You might have to do manual changes to the HKEY_LOCAL_MACHINE path in order to find the tools from any other directory.

12.2.1 Toc2Bkx.EXE

The **Toc2BKX.EXE** is used to convert an ASCII file with "*table of contents data*" (typically a cut & paste from a table of contents in a PDF file) into a bookmark index file (.BKX) that can be imported into the PDF file. The typical use case is shown in 7.1.5.3 "Case 3: No bookmarks in PDF, but Table of Contents" on page 7-94.

12.2.2 Bkx2Stb.EXE

The **Bkx2Stb.EXE** is used to generate stub files from an existing bookmark file. The typical use case is shown in 7.1.5.6 "Case 6: SECURED document - no bookmarks nor TOC" on page 7-99.,

12.2.3 SetEnvReg.EXE

SetEnvReg.EXE is *the command line version* of the Set/Clear documentation environment function of the Adobe Acrobat plug-in.

Invokation is done by

```
D:\>SetEnvReg  <InstallPath>
```

It is used to set the documentation environment, typically used in a batch file. It performs exactly the same function as the plug-in (→ 11.7 "Set/Clear documentation environment" on page 11-178.

Example

If you have your stubs to be placed under "**S:\SkyLine\Documentation\dev\ref\stb**" then your project environment **InstallPath** would be "**S:\SkyLine\Documentation**". the "dev\ref\stb" subdirectories are fixed and shall not be changed – not at last this constraint helps to keep this documentation in line with the actual installations. That aspect is described in 11.7.1 "Setting the documentation environment".

12.2.4 AcroDDE.EXE

The ND.API plug-in is only to work with commercial *Acrobat products* and will mostly be used by the Editorial Team only.

Another **AcroDDE.EXE** was developed to allow using the Stub management with the *Adobe Reader*. As a matter of fact, Stub management is the only function that is important to a reader whereas the plug-in functions are mostly related to document *creation*, not reading.

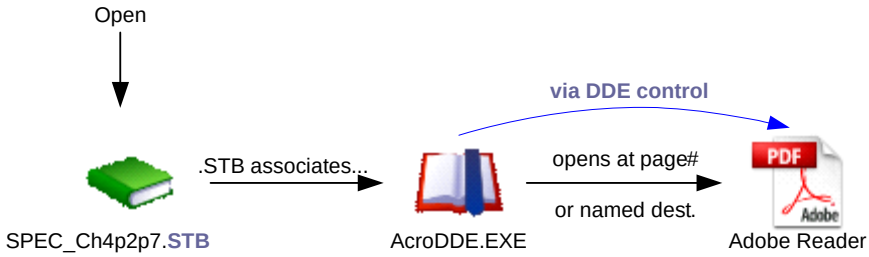


Figure 12-33. How AcroDDE works

Whereas with the ND.API plug-in the open procedure could be extended directly in the Acrobat Reader, AcroDDE.EXE is a seperate program to interprete stub content and open a PDF file in Adobe Reader. Also see Figure 12-14 on page 12-198.

The DDE interface is described in [AdobeDDE#3] and [AdobeAPI#2].

12.2.4.1 Using AcroDDE.EXE

The command syntax for using *AcroDDE.exe* from a command line prompt is

```
acrodde MyFile.{stb|pdf}   [<nameddest>]
```

Acrobat installed

With an installed Acrobat Reader, *acrodde* can directly launch the PDF file and target the named destination, as well as to address via Stub management.

Adobe Reader only

With an installed *Adobe Reader only*, *acrodde* can only open the PDF but cannot go to the *named destination* - Adobe Reader does not allow to control through DDE access for more than the page. Opening a stub file will allow to go to the *page* where the named destination is coded.

If Adobe Reader only is installed, the installation program 01_InstND.EXE recognizes the situation and associates STB files directly to *AcroDDE.EXE* to be interpreted. The ND.API plug-in cannot be installed on Adobe *Reader*, (due to Adobe's intentional restriction on the free product) - therefore everything has to go through the *AcroDDE.EXE* → "Installing AcroDDE.EXE".

12.2.4.2 Installing AcroDDE.EXE

If Acrobat Software (Acrobat Standard / Professional) is not being used but the Adobe Reeder is installed (Freeware) then the ND.API plug-in cannot be installed To install a plug-in for the Adobe Reader, Adobe Corp. hast to sign the plug-in at a yearly fee of around $1000. Since the Adobe Reader does not offer all the functions allowed with the Acrobat version, we choose another way.

AcroDDE.EXE will be associated to the *.STB* extension and do redirection as described in 12.1.18 "Stub management" on page 12-200. However technically this is realized through DDE control.

If Adobe Reader is found (and no Acrobat installed), then the InstNd.EXE copies AcroDDE.EXE and AcroDDE.ICO into the installation directory of the Adobe Reader and associates the *.STB* extension to the AcroDDE.EXE.

The manual installation of AcroDDE.EXE is described in 11.3.2 "Associating to AcroDDE.EXE".

12.2.5 01_InstND.EXE

01_InstND.EXE is used to install the The ND.API plug-in or the AcroDDE.EXE. The following chapters explain more detail. In general InstND.EXE tries to find the installation path for the Acrobat Reader, if not found it checks the installation path for the Adobe Reader. If such a path is found, either the ND.API (Acrobat product found) is installed or the AcroDDE.EXE (only Adobe Reader found) is installed.

The 01_InstND.EXE also installs the Accompanying tools which are required for some special cases → 7.1.5.5 “Case 5: SECURED document with TOC but no Bookmarks”.

The tools are command line tools which allows them to be used in Batch files. To open a command window use the Windows Start button and enter "cmd" in the search field.

PATH variable

Installing with 01_InstNd.EXE automatically sets the user PATH= variable to the tools, however if the user defintion addes to the end of the HKEY_LOCAL_MACHINE defined path, then the entire path may easily extend 256 bytes and will no more be recognized by the command prompt.

You might have to do manual changes to the HKEY_LOCAL_MACHINE path in order to find the tools from any other directory.

12.2.5.1 Installing the ND.API

The ND.API plug-in is an extension to the Adobe Acrobat (Standard/Professional) program. Having this plug-in installed adds a new menu entry "ND-xtreme" which allows to handle named destinations (and do much more!). The ND.API plug-in is installable if you use Adobe Acrobat (Standard/Professional) version. 7.x to 9.x and higher.

> The *installation path of the Adobe Acrobat (Reader)* contains a directory named “plug-ins”. Placing (copying) a plug-in (<*my_pluging.API*>) to this directory is enough to let the Acrobat Reader pick up the functionality.

The **01_InstND.EXE** copies the ND.API into the right place. ND.API shall be in the same directory as InstND.EXE. Typically they are found under *%ProgramData%\ezread\AdobePlugin*.

To install: click on **01_InstNd.EXE**. The ND.API plug-in is copied into the Acrobat “plug-ins” directory and will be found on the next start.If it fails

If the program fails, most times, an instance of the Adobe Acrobat is running while trying to install. A more seldom error case is, that neither Acrobat nor the freeware Adobe Reader are installed.

Technical description: **01_InstNd.EXE** checks the registry for the existence of Acrobat seeking `HKEY_CURRENT_USER\Software\Adobe\Adobe Acrobat\<version>\InstallPath` or alternatively `HKEY_LOCAL_MACHINE\Software\Adobe\Adobe Acrobat\<version>\InstallPath` in the registry and installs “The ND.API plug-in” in the plug-ins directory of the Acrobat installation. Acrobat shall be closed during this operation.

Associate STB

The *.STB* extension will be associated to the Acrobat Reader (matching the PDF definition) therefore an *.STB* file will open the Acrobat like a *PDF* file does.

12.2.6 02_AutoHotkey_L_Install.exe

This program is a sophisticated key macro program and a very practical solution to efficient work. It is freeware and a very powerful tool to make the work faster. The installation is done from the *%ProgramData%\ezread\AdobePlugin* path.

12.2.6.1 Basic HotKey script – 03_Basic.ahk

The following source is a basic hotkey script useable to realize the Instant Jump to Stub function

```
;--------------------------------------
; Instant launch of a document using F12 technology
;--------------------------------------
^+!F12::
AutoTrim, Off
InputBox, DocuTag, Enter document tag, Example: ezRead#3.3 ,, 200, 140
if ErrorLevel
  return
BracketTag = [%DocuTag%]
; MsgBox BracketTag = %BracketTag%

RegRead, ProjDoc, HKEY_CURRENT_USER, Environment, ProjDoc

StbStr  := ChangeBookMark(BracketTag)
PosExt  := Instr(StbStr, "_")
PathExt := SubStr(StbStr, 1, PosExt-1)
StbPath = %ProjDoc%\dev\ref\stb\%PathExt%\%StbStr%.stb
IfExist %StbPath%
  Run, %StbPath%
  Clipboard =
  ClipStr  := RegExReplace(StbStr, "\\", "/", 0, -1, 1)
  Clipboard = %StbPath%
AutoTrim, On
return
;=====================================
; Functions
;=====================================
ChangeBookMark(ClipText)
{
; remove "["
  LeftBrk := InStr(ClipText, "[")
  StringTrimLeft, StRi, ClipText, LeftBrk

; remove "]"
  RightBrk := InStr(StRi, "]")
  CutRight := StrLen(StRi) - RightBrk + 1
  StringTrimRight, StLe, StRi, CutRight
;MsgBox RightBrk = %StLe%

; replace Prefix dot by version 'v' or point 'p' if part of a text with no numbers
  DotStr  := RegExReplace(StLe, "_", "u", 0, -1, 1)
  ChStr   := RegExReplace(DotStr, "(?<=[0-9])\.(?=.*#)","v")
  StLe    := RegExReplace(ChStr, "(?<![0-9])\.(?=.*#)","p")

; replace chapter delimiter #
  ChStr   := RegExReplace(StLe, "#(?=[0-9])","_Ch")
  StLe    := RegExReplace(ChStr, "#(?![0-9])","_")

; replace general space, dot, dash
  ChStr   := RegExReplace(StLe, " ", "", 0, -1, 1)
  DotStr  := RegExReplace(ChStr, "\.", "p", 0, -1, 1)
  DashStr := RegExReplace(DotStr, "-", "s", 0, -1, 1)
  StbStr  = %DashStr%
; MsgBox StbStr = %StbStr%
  return %StbStr%
}
```

Figure 12-34. Basic HotKey Script to use Instant Jump to Stubs

12.2.7 ClipText.EXE

This file is helpful when copying text from a PDF or a any formatted file into your present text editor. Typically the formatting information (e.g. Bold, Font) is copied together with the text. This is not always wanted, instead the raw text is desired to be taken in.

Using ClipText.EXE in combination with the 02_AutoHotkey_L_Install.exe will remove any formatting information and leave the raw ASCII code in the clipboard. This result can then be easily pasted into the present editor

```
;--------------------------------------
; Clipboard commands - ezRead
;--------------------------------------
; Convert Clipboard to pure ASCII
^+Ins::
Send ^C
RegRead, ProjTools, HKEY_CURRENT_USER, Environment, ProjTools
Run, %ProjTools%\ClipText.exe,,Hide
return

; Convert Clipboard to pure ASCII and remove CRLF
^+Del::
Send ^C
RegRead, ProjTools, HKEY_CURRENT_USER, Environment, ProjTools
Run, %ProjTools%\ClipJoin.exe,,Hide
return
```

Figure 12-35. HotKey Script to use Clipboard functions

The suggested script assigns ClipText.EXE to the **CTRL + SHIFT + Ins** key.

12.2.8 ClipJoin.EXE

This EXE does mainly the same as ClipText.EXE, however on copying it replaces any occurrence of a single CRLF into a space. The function is helpful when taking in text from a PDF file. A PDF file does not have an evidence about the actual line breaks of the original source – instead it creates a line break (CRLF) wherever a new line is detected by the location of words.

Copying from a PDF will therefore contain undesired line breaks which will be removed by the ClipJoin.EXE. Of course you will have to re-format the pasted paragraph and add some line breaks. It has, however, been shown, that this 'additional' work is much faster to do, than visiting many undesired line breaks and deleting them manually.

The suggested script in Figure 12-35 assigns ClipText.EXE to the **CTRL + SHIFT + Del** key.

13 Glossary and Abbreviations

13.1 Abbreviations

AHW	Hardware Architecture
ASW	Software Architecture
ASY	System Architecture
CRQ	Change Request
CRS	Customer Requirements Specification
CRT	Certification documents
DMA	Direct Memory Access
ECC	European Citizen Card
EMC	Electromagnetic Compatibility
IMP	Implementation documentation
REQ	Requirements documentation
SBD	Storyboard
STARS	→ 2.3.3 “Stakeholder Requirement Specification (STARS)” on page 2-17
SysRS	→ 2.3.4 “System Requirement Specification (SysRS)” on page 2-17
TOE	Target of evaluation
TST	Test architecture
VISIO	MicroSoft Visio Drawing Tool
UML	Unified Modeling Language
URS	User Requirements Specification

13.2 Glossary

A

Application Specific use of functional interfaces provided by a generic system

ARC42 A template for architecture documents from Dr. Peter Hruschka und Dr. Gernot Starke. The current version is 3.2 (January 2008). It is available under the Creative Common Licence, which allows free usage even in commercial environment. For further information, see http://www.arc42.de/template/template.html.

Architecture Description of composition and interaction of functional units (components) in a specified context.

AURS (deprecated) Answer to User Requirements Specification (→ URS)

The AURS is written by the development group as a specification of what and how the requirements in the URS can be met. The AURS typically contains raw design propsals and corresponding cost estimates in order to allow a discussion between ordering dept and development on the most appropriate realization. This process may iterate the URS as well as the AURS to finally match the requirements and their realization.

G&D does no more support the concept of URS/AURS but replaced it by the more efficient STARS.

Authentication Authentication is the act of establishing or confirming something (or someone) as authentic, that is, that claims made by or about the thing are true. This might involve confirming the identity of a person, the origins of an artifact, or assuring that a computer program is a trusted one.

http://en.wikipedia.org/wiki/Authentication

C

CA In cryptography, a certificate authority or certification authority (CA) is an entity which issues digital certificates for use by other parties. It is an example of a trusted third party. CAs are characteristic of many public key infrastructure (PKI) schemes. There are many commercial CAs that charge for their services. There are also several providers issuing digital certificates to the public at no cost. Institutions and governments may have their own CAs.

http://en.wikipedia.org/wiki/Certification_authority

CC EALn The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (IEC 15408) for computer security.

Common Criteria is based upon a framework in which computer system users can specify their security requirements, vendors can then implement and/or make claims about the security attributes of their products, and testing laboratories can evaluate the products to determine if they actually meet the claims. In other words, Common Criteria provides assurance that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous and standard manner.

http://en.wikipedia.org/wiki/Common_criteria

Component Functional unit with a specified interface

CSV

Comma separated value. A method to separate fields in an ASCII file.
Every line of the ASCII file represents the row of a table or a data record The fields are separated by commas.

Example: `VIP,19891022,”Scherzer, Anne-Christine”,””Anki””`

contains the 4 fields: [VIP] [19891022] [Scherzer, Anne-Christine] [“Anki”]

To express a comma within the field, the entire text of the field is put in quotes “ “. Hence a comma within the quotes will not be interpreted as field separator.

To express a quote “-character, *double quotes* will be used

D

DDE

Dynamic Data Exchange : A mechanism to allow inter-application-communication. If an application provides a DDE server functionality, another application may address this server and exchange messages.

The Adobe Reader prodcut, for instance, allows some functions to be executed via DDE e.g. “Open Document at page ..” or Open Document at Named Destination”.

DOORS

DOORS requirement management tool

DÜESS

a basic format description of a personalization order and the official verb for the entire idea of a personalizationi description.

E

EMF

→ WMF

EPS

Encapsulated PostScript, or EPS, is a DSC-conforming PostScript document with additional restrictions intended to make EPS files usable as a graphics file format.

In other words, EPS files are more-or-less self-contained, reasonably predictable PostScript documents that describe an image or drawing, that can be placed within another PostScript document.

http://en.wikipedia.org/wiki/Encapsulated_PostScript

F

Feature

Behavior of a basic system that fulfills a specified use case

Fine architecture

Architecture that provides a complete an sufficient description of components, mechanisms and interfaces such that an unmistakable implementation of the described interfaces and function is possible.

FIPS

Federal Information Processing Standards (FIPS) are publicly announced standards developed by the United States Federal government for use by all non-military government agencies and by government contractors.

Many FIPS standards are modified versions of standards used in the wider community ANSI, IEEE, ISO, etc.)

http://en.wikipedia.org/wiki/Federal_Information_Processing_Standard

Framework

Sufficient specification of mandatory interfaces and rules within which a variety of implementations is possible and allowed.

O

OLE

Object Linking and Embedding (OLE) is a technology that allows embedding and linking to documents and other objects developed by Microsoft. For developers, it brought OLE custom controls (OCX), a way to develop and use custom user interface elements.

For example, a *desktop publishing system* might send some text to a *word processor* or a picture to a *bitmap editor* using OLE. The main benefit of using OLE is to display visualizations of data from other programs that the host program is *not normally able to generate itself* (e.g. a pie-chart in a text document), as well as to create a master file.

http://en.wikipedia.org/wiki/Object_Linking_and_Embedding

Operating System

Software providing at least interfaces and associated functionality in order to manage the hardware variants of possible computing systems

P

Platform

System providing a basic behavior and a set of interfaces and associated functions in order to implement additional use cases or products by using these provisions.

PNG

Portable Network Graphics (PNG) is a bitmapped image format that employs lossless data compression. PNG was created to improve upon and replace GIF (Graphics Interchange Format) as an image-file format not requiring a patent license.

PNG supports palette-based (palettes of 24-bit RGB colors), greyscale or RGB images. PNG was designed for transferring images on the Internet, not professional graphics, and so does not support other color spaces (such as CMYK). The major advantage of PNG graphics is that they support transparency.

http://en.wikipedia.org/wiki/Portable_Network_Graphics

Product

A physical thing or a service which is subject to trade and legal evidence in commercial context

R

Raw architecture

Architecture that only focuses on a minimum of components, functions and interfaces necessary in order to understand the basic function of the context or target system

S

T

Toolbox

Set of components with sufficiently specified interfaces to allow the combination of these components to form a functional unit whose purpose is not predictable by the components itself

U

URL

In computing, a Uniform Resource Locator (URL) is a Uniform Resource Identifier (URI) which also specifies where the identified resource is available and the protocol for retrieving it.

In popular usage and in many technical documents it is often confused as a synonym for *uniform resource identifier*.

Every URL begins with the scheme name that defines its namespace, purpose, and the syntax of the remaining part of the URL. Most Web-enabled programs will try to dereference a URL according to the semantics of its scheme and a context-vbn. For example, a Web browser will usually dereference the URL http://example.org/ by performing an HTTP request to the host *example.org*, at the default HTTP port (port 80).

Dereferencing the URL *mailto:bob@example.com* will usually start an e-mail composer with the address *bob@example.com* in the *To field*.

example.com is a domain name; an IP address or other network address might be used instead. In addition, URLs that specify https as a scheme (such as *https://example.com/*) normally denote a *secure* website.

http://en.wikipedia.org/wiki/URL

URS
(deprecated)

User Requirement Specification, a document which belongs to project documentation specifying the raw function and non-functional requirements of the product.

The URS shall be descriptive enough in order to allow a cost/PM estimation for the development/testing with focus on a defined exit level (prototype leve, product level etc.).

G&D does no more support the concept of URS/AURS but replaced it by the more efficient STARS.

W

WMF

Windows Metafile (WMF) is a graphics file format on Microsoft Windows systems, originally designed in the early 1990s. Windows Metafiles are intended to be portable between applications and may contain both vector and bitmap components. In contrast to raster formats such as JPEG and GIF which are used to store bitmap graphics such as photographs, scans and graphics, Windows Metafiles generally are used to store line-art, illustrations and content created in drawing or presentation applications. Most Windowsclipart is in the WMF format.

http://en.wikipedia.org/wiki/Enhanced_Metafile

14 INDEX

A

Abbreviations 65
Acceptance criteria 149
Accompanying documentation 44
Accuracy 23
Acrobat Reader 215
AcroDDE.EXE 214, 215
Adobe
  Acrobat 215
  Reader 214, 215
Adobe Reader
  Change Security Settings 174
  Working with 47
Apache 60
Architecture 56
  Aspects 61
  criteria 151, 157
  Software 20, 39
  System 38
  Test 43
  Writing 51
Auto-Generation 41

B

Blackbox 52, 56, **58**
Bookmarks
  creating from Table of Contents 94
  CSV file structure 194
  enumerate 93, 105, 196
  export 194
  generate from named destinations 188
  import 96, 194
  magnetic connect 194
  Settings 136
Books
  exporting 110
  final release 80
  Organization 35
  using 26
  Where to store 70
Building Block 52, 56
  criteria 152

C

Certification 43
certification
  documents 43
  report 44
Cocoon
  criteria 161
Coding conventions 44
Context View 55
  criteria 152
Cross-References 26, 64, 85

D

DDE 174
Dependencies 59
Deployment View 60
Descriptions 42
Design decisions 60
Directories 70, 178
Document
  consistency 147
  exporting link trees 110
  Quality 63
  SECURED 97, 200
  storing 70, 139
  Structure 54
Documentation
  Architecture 18
  environment 178
  Excellence 20, 21
  External 65
  release 80
  Requirements 21
  Rules 28, 51
  storing 139
  Strategy 26
  Understandability criteria 150
  User 20
  verification 147
Documentation Guide 44
DOORS 37

E

Editorial Team 27
Enterprise Architect 67
Enumerate bookmarks 196
eq Tag 77
Evaluation
  documents 44
ezRead
  Name 8
exmp Tag 78
Export
  Book Files 110
Export Format
  Enterprise Architect 67
External documentation 65

F

Figures 66
Files
  Where to store 70
Fonts
  installing 176
FormatRef.FM 67
Formatting Templates 67

G

Glossary 65
Graphics 66
  Import 66
  Time stamp 66
  VISIO 66

H

Hardware Architecture
  Architecture
    Hardware 40
Highlighting 64
Hyperlink 27, 64, 85
  MS-WORD 120

I

Image
  Import 66
Implementation Documentation 40
Import Graphics 66
Index 61
Installation
  Fonts 176
  ND.API 175
  Tools 163

M

M8.newlink 91
Magnetic connect feature 194
Maintenance 23
Media 24
Meta-Tags 28, 72
  eq 77
  ND 78
  tab 76
  url 74
  v Tag 74
  xmp, exmp 78
  xref 72
MicroSoft WOrd 119
Modification Date 66
Modularity 24
MS-WORD 106, 119
  artifacts 128
  as default editing tool 119
  Bad page conversion to PDF 134
  conversion filter 119
  converting links 132, 208
  create bookmarks for PDF 122
  generate hyperlinks 119
  Hyperlink 120
  Print to PDF 124
  Save as PDF 124
  unwanted artifacts 128

N

Named destinations 65, 80, 85
  bookmark generation 188
  clearing 187
  create automatically 92
  creating manually 88
  CSV file structure 185
  export and import 185
  viewing in PDF 86
Navigation 47
ND Tag 78
ND.API 183
  Installation 175
Notations 47

P

Paragraph
  Style 64
PDF
  Export Format 29
PDF conversion
  Bad page conversion in MS-WORD 134
PDF format 47
plug-in 183
Prefix
  Document name prefix 142
Project Documentation 45

Q

Quality Goals 21

R

Reference Documentation
  storing 139
Reference documentation 45
Release documentation 80
Requirements 36, 39
  Verification 25
Rules
  verification 147
Runtime View 60

S

Screenshots 66, **67**
SECURED documents 97
SetEnvReg.EXE utility 214
Software
  Design 18
Software Architecture 39
Software architecture 20
Storyboard 44
Structure Template 52
Stubs 97
  Technique 200
Style
  Highlighting 64

Paragraph 64
System architecture 38
Architecture
System 19
System variables 69
SysVars.FM 69

T

tab Tag 76
Templates
formatting 67
Test
Architecture 43
Test Architecture 43
Time stamp 66
Toc2Bkx utility 214
Tools
install 163
SetEnvReg 214
Toc2Bkx 214

U

UML 67
url Tag 74
User documentation 20

V

v Tag 74
Vector based format 67
Verification 147
priorities 147
Version 150
Version control 161
Version management 81
View
Deployment 60
Runtime 60
Views 52
VISIO 66

W

Whitebox 52, 56
Windows 7 165
Writing Style 64

X

XSLT
criteria 161

Z

xmp Tag 78
xref 72