EVIDEN

Identity and Access Management

DirX Identity

WebCenter Customization Guide

Version 9.0.2, Edition November 2026



All product names quoted are trademarks or registered trademarks of the manufacturers concerned.

© 2026 Atos Group

All Rights Reserved

Distribution and reproduction not permitted without the consent of Atos Group.

Table of Contents

Copyright ii
Preface 1
DirX Identity Documentation Set 2
Notation Conventions 4
1. Getting Started 6
1.1. Deploying Web Center 6
1.1.1. Deploying Web Center with the DirX Identity Configuration Program 6
1.1.2. Deploying Web Center Manually 7
1.1.2.1. Deploying Additional Web Center Instances 8
1.2. Preparing the Development Environment 8
2. Customization Files 10
2.1. Properties Files 10
2.1.1. Web Center Properties 10
2.1.2. Paths 10
2.2. Message Texts 11
2.3. Objects Configuration 12
2.4. Renderers 13
2.4.1. Renderer Definitions 13
2.4.2. Default Renderer Assignments 14
2.5. Including Custom Javascript Files and Stylesheets 15
2.6. Javascript 16
2.6.1. Custom Code 16
2.6.2. Custom Renderers 16
2.6.3. Custom Validators 16
2.6.4. Custom Messages 16
2.6.4.1. Customization Sample 17
2.6.4.1.1. Default Message File 17
2.6.4.1.2. Custom Message File 17
2.7. Custom File Overview 18
2.8. Configuring Customization File Names 19
3. Customization Examples 21
3.1. Changing the JSP Page Layout 21
3.1.1. Common Layout Template layout.jsp 21
3.1.2. CSS Styles 22
3.1.2.1. Form Styles (styles.css) 22
3.1.2.2. Page Styles (styles.css) 23
3.2. Adding Languages 24
3.2.1. Content of a Language Directory 24
3.2.2. Using Language-Dependent Texts 24

3.2.3. Using Text Files . . . 25
3.2.4. Adding a New Language . . . 26
3.3. Changing Attribute Lists in Pages . . . 26
3.4. Removing Parts of the Default Application . . . 26
3.4.1. Customizing the Context Configuration webCenter-*domain*.xml . . . 26
3.4.2. Struts and Forms Configuration . . . 27
3.4.3. Tiles and Menu Definitions . . . 28
3.5. Integrating Management of a New Object Type . . . 28
3.5.1. Creating Separate Configuration Files . . . 28
3.5.2. Searching Contracts . . . 29
3.5.2.1. Struts-Config.xml . . . 29
3.5.2.2. Tiles-defs.xml . . . 31
3.5.2.3. Forms-config.xml . . . 33
3.5.3. Creating Menus and Static HTML Pages . . . 35
3.5.3.1. Creating a Menu for Contract Processing . . . 35
3.5.3.2. Creating Contract Context Menus . . . 35
3.5.3.3. Extending the Message Files . . . 36
3.5.3.4. Creating Static HTML Texts . . . 37
3.5.4. Extending the Navigation History Control . . . 37
3.5.5. Displaying Contract Data . . . 37
3.5.5.1. Definitions in forms-config.xml . . . 38
3.5.5.2. Definitions in struts-config.xml . . . 39
3.5.5.3. Definitions in tiles-defs.xml . . . 39
3.5.6. Modify a Contract . . . 40
3.5.6.1. Definitions in struts-config.xml . . . 40
3.5.6.2. Definitions in tiles-defs.xml . . . 42
3.5.6.3. Definitions in forms-config.xml . . . 43
3.5.7. Creating a Contract . . . 43
3.5.7.1. Definitions in struts-config.xml . . . 44
3.5.7.2. Definitions in tiles-defs.xml . . . 45
3.5.7.3. Definitions in forms-config.xml . . . 45
3.5.8. Configuring Access Policies . . . 46
4. Renderers . . . 47
4.1. Overview . . . 47
4.2. Custom File Overview . . . 47
4.3. Customizing Renderers . . . 49
4.4. Creating a New Renderer . . . 49
4.4.1. Sample Renderer Use Case . . . 49
4.4.2. Renderer Definition . . . 50
4.4.3. Code Snippets . . . 51
4.4.3.1. HTML Form Field - Read-Only Mode . . . 51
4.4.3.2. HTML Form Field - Editable Mode . . . 52

4.4.3.3. HTML Table Cell . . . 52
4.4.4. Javascript Functions . . . 52
4.4.4.1. Scriptlet Function . . . 53
4.4.4.2. Table Cell Renderer Function . . . 53
4.4.4.3. Registering Renderers . . . 56
4.4.4.4. Making the Code Available to the Client . . . 56
4.4.5. Message Texts . . . 57
4.4.5.1. English . . . 57
4.4.5.2. German . . . 57
4.4.6. Stylesheets . . . 57
4.4.6.1. Medium Font Size . . . 57
4.4.6.2. Large Font Size . . . 58
4.4.6.3. Making the Styles Available to the Client . . . 59
4.5. Extending Existing Renderers . . . 59
4.5.1. Reassigning Property Values . . . 59
4.5.2. Reassigning Code Snippets . . . 60
4.6. Assigning Renderers to Properties . . . 60
4.6.1. Assigning a Renderer to Specific Form Fields . . . 60
4.6.1.1. Form Property . . . 60
4.6.1.2. Data Property . . . 61
4.6.2. Assigning a Renderer as Default Renderer . . . 61
4.6.2.1. Property Name . . . 61
4.6.2.2. Editor . . . 62
4.6.2.3. Property Type . . . 62
4.7. Renderer Samples . . . 62
4.7.1. Renderer Definitions . . . 62
4.7.1.1. WEB-INF/custom/config/renderers-config.xml . . . 62
4.7.2. Renderer Snippets . . . 63
4.7.2.1. WEB-INF/custom/snippets/textField/italicTextField.js . . . 63
4.7.2.2. WEB-INF/custom/snippets/textField/redTextField.js . . . 63
4.7.2.3. WEB-INF/custom/snippets/textField/roRedTextField.htm . . . 63
4.7.2.4. WEB-INF/custom/snippets/textField/rwRedTextField.htm . . . 63
4.7.3. Javascript Code . . . 63
4.7.3.1. resources/custom/scripts/custom.js . . . 64
5. Validation . . . 65
5.1. Overview . . . 65
5.2. Validators . . . 66
5.2.1. Single-Property Validators . . . 66
5.2.1.1. Predefined Validators . . . 66
5.2.1.1.1. fax . . . 66
5.2.1.1.2. mail . . . 66
5.2.1.1.3. print . . . 67

5.2.1.1.4. integer . . . 67
5.2.1.1.5. date . . . 67
5.2.1.1.6. time . . . 67
5.2.1.2. Sample Custom Validators . . . 67
5.2.1.2.1. customEmployeeNumber . . . 67
5.2.1.2.2. customUid . . . 68
5.2.1.2.3. customPhone . . . 69
5.2.1.2.4. customPrime . . . 71
5.2.2. Multiple-Property Validators . . . 72
5.2.2.1. Predefined Validators . . . 73
5.2.2.1.1. minMax . . . 73
5.2.2.1.2. dates . . . 73
5.2.2.1.3. times . . . 74
5.2.2.2. Sample Custom Validators . . . 74
5.2.2.2.1. customEmployee . . . 74
5.2.3. Registering Validators . . . 75
5.3. Validation Renderers . . . 76
5.3.1. Single-Property Renderers . . . 76
5.3.1.1. Base Renderers . . . 76
5.3.1.1.1. formattedText . . . 76
5.3.1.1.2. integer . . . 77
5.3.1.1.3. boundedNumber . . . 77
5.3.1.1.4. lowerBoundedNumber . . . 78
5.3.1.1.5. calendar . . . 78
5.3.1.1.6. boundedDate . . . 78
5.3.1.1.7. upperBoundedDate . . . 79
5.3.1.1.8. time . . . 79
5.3.1.1.9. timeWithSeconds . . . 80
5.3.1.2. Predefined Renderers . . . 81
5.3.1.2.1. fax . . . 81
5.3.1.2.2. mail . . . 81
5.3.1.2.3. mailList . . . 81
5.3.1.2.4. phone . . . 81
5.3.1.2.5. naturalNumber . . . 81
5.3.1.2.6. nonNegativeInteger . . . 82
5.3.1.2.7. integerList . . . 82
5.3.1.3. Sample Custom Renderers . . . 82
5.3.1.3.1. customEmployeeNumber . . . 82
5.3.1.3.2. customEmployeeNumberList . . . 83
5.3.1.3.3. customUid . . . 83
5.3.1.3.4. customPhone . . . 83
5.3.1.3.5. customPhoneList . . . 83

5.3.1.3.6. customAge . . . 84
5.3.1.3.7. customPrime . . . 84
5.3.1.3.8. customPrimeList . . . 84
5.3.1.3.9. custom21stCentury . . . 84
5.3.1.3.10. customOneYear . . . 84
5.3.2. Multiple-Property Renderers . . . 85
5.3.2.1. General Renderer Properties . . . 85
5.3.2.1.1. ids . . . 85
5.3.2.1.2. tooltips . . . 85
5.3.2.2. Base Renderers . . . 85
5.3.2.2.1. minMax . . . 85
5.3.2.2.2. boundedMinMax . . . 86
5.3.2.2.3. boundedMinMaxLess . . . 86
5.3.2.2.4. lowerBoundedMinMax . . . 87
5.3.2.2.5. lowerBoundedMinMaxLess . . . 87
5.3.2.2.6. checkedDate . . . 87
5.3.2.2.7. checkedTime . . . 88
5.3.2.3. Predefined Renderers . . . 88
5.3.2.3.1. startEndDate . . . 88
5.3.2.3.2. startEndDateLess . . . 89
5.3.2.3.3. startEndDateWithDisabled . . . 89
5.3.2.3.4. startEndDateWithDisabledLess . . . 90
5.3.2.4. Sample Custom Renderers . . . 90
5.3.2.4.1. customPasswordLength . . . 90
5.3.2.4.2. customDepartureArrival . . . 91
5.3.2.4.3. customEmployee . . . 91
5.3.2.4.4. customStartEndTime . . . 91
5.4. Assigning Renderers to Properties . . . 92
5.4.1. Assigning Renderers to Individual Form Fields . . . 92
5.4.1.1. Single-Property Renderers . . . 92
5.4.1.2. Multiple-Property Renderers . . . 94
5.4.1.3. Combining Single-Property and Multiple-Property Renderers . . . 94
5.4.2. Assigning Validation Renderers as Default Renderers . . . 95
5.5. Customization Hints . . . 95
5.5.1. Javascript Code . . . 95
5.5.2. Tooltips . . . 96
5.5.3. Renderers . . . 96
5.6. Validator Samples . . . 96
5.6.1. Javascript Code . . . 97
5.6.1.1. resources/custom/scripts/custom.js . . . 97
5.6.2. Tooltips . . . 98
5.6.2.1. resources/custom/scripts/messages/en/messages.js . . . 98

6. Photos and Certificates . . . 100
6.1. Prerequisites . . . 100
6.2. Renderers . . . 101
6.2.1. Photo . . . 101
6.2.1.1. Form Field Configuration Sample . . . 101
6.2.1.2. Renderer Configuration . . . 101
6.2.1.2.1. Attributes . . . 101
6.2.1.2.2. Properties . . . 101
6.2.1.2.3. Sample . . . 102
6.2.1.3. HTML/Javascript Rendering . . . 102
6.2.2. PhotoDownload . . . 102
6.2.2.1. Form Field Configuration Sample . . . 102
6.2.2.2. Renderer Configuration . . . 102
6.2.2.2.1. Attributes . . . 103
6.2.2.2.2. Properties . . . 103
6.2.2.2.3. Sample . . . 103
6.2.2.3. HTML/Javascript Rendering . . . 103
6.2.3. Photos . . . 104
6.2.3.1. Form Field Configuration Sample . . . 104
6.2.3.2. Renderer Configuration . . . 104
6.2.3.2.1. Attributes . . . 104
6.2.3.2.2. Properties . . . 104
6.2.3.2.3. Sample . . . 105
6.2.3.3. HTML/Javascript Rendering . . . 105
6.2.4. Documents . . . 105
6.2.4.1. Sample . . . 105
6.2.5. Certificate Download . . . 106
6.2.5.1. Form Field Configuration Sample . . . 106
6.2.5.2. Renderer Configuration . . . 106
6.2.5.2.1. Attributes . . . 106
6.2.5.2.2. Properties . . . 106
6.2.5.2.3. Sample . . . 107
6.2.5.3. HTML/JavaScript Rendering . . . 107
6.2.6. Certificates . . . 107
6.2.6.1. Form Field Configuration Sample . . . 107
6.2.6.2. Renderer Configuration . . . 108
6.2.6.2.1. Attributes . . . 108
6.2.6.2.2. Properties . . . 108
6.2.6.2.3. Sample . . . 108
6.2.6.3. HTML/JavaScript Rendering . . . 109
6.3. Java Renderer Classes . . . 109
6.3.1. DownloadRenderer . . . 109

6.3.1.1. Properties . . . 110
6.3.1.2. Samples . . . 110
6.3.2. X509CertificateRenderer . . . 111
6.3.2.1. Properties . . . 111
6.3.2.2. Certificate Field Names . . . 112
6.4. Configuration in web.xml . . . 113
6.4.1. The Binary Reader Servlet . . . 113
6.4.1.1. Definition . . . 113
6.4.1.2. Mappings . . . 114
6.4.2. The Binary Request Filter . . . 114
6.4.2.1. Definition . . . 114
6.4.2.2. Mappings . . . 115
6.4.3. The AddHeaderFilterForDownloads Filter . . . 115
6.4.3.1. Definition . . . 115
6.4.3.2. Mappings . . . 116
6.5. Approving Photos and Certificates . . . 116
6.5.1. Modification Approval . . . 116
6.5.2. Creation Approval . . . 117
7. DN Representation . . . 119
7.1. Overview . . . 119
7.1.1. Terms . . . 119
7.2. Configuration . . . 119
7.2.1. Client-Side Configuration . . . 120
7.2.1.1. Configuration Parameters . . . 120
7.2.1.2. Parameter includeNodes . . . 120
7.2.1.3. Parameter Evaluation Order . . . 121
7.2.1.4. Samples . . . 121
7.2.2. Server-Side Configuration . . . 122
7.2.2.1. Configuration Parameters . . . 122
7.2.2.2. Sample . . . 122
7.2.3. Renderer Configuration . . . 123
7.2.3.1. Affected Renderers . . . 123
7.2.3.2. Renderer Parameter . . . 123
7.2.3.3. Sample . . . 123
7.2.3.4. Form Property Configuration . . . 124
7.2.3.5. Sample . . . 124
7.3. Individual DN Representations . . . 124
7.3.1. Single-valued DN attributes . . . 124
7.3.1.1. Samples . . . 125
7.3.1.1.1. Form Property Definition . . . 125
7.3.1.1.2. Data Property Definition . . . 125
7.3.2. Multi-valued DN attributes . . . 125

7.3.2.1. Samples . . . 125
7.3.2.1.1. Form Property Definition . . . 126
8. Attribute Modification Approval . . . 127
8.1. Overview . . . 127
8.2. Form Configuration . . . 127
8.2.1. Base Form Configuration . . . 127
8.2.1.1. Top and Bottom . . . 128
8.2.1.2. Includes . . . 128
8.2.2. Included Forms . . . 129
8.2.2.1. Section . . . 129
8.2.2.2. Attributes . . . 129
8.2.3. Generic Attribute Display . . . 130
8.3. Restrictions . . . 130
8.4. Notes . . . 131
9. Localizing Properties with a Finite Number of Values . . . 132
10. Customizing the Overall Page Layout . . . 134
10.1. Files . . . 134
10.2. Layout Properties . . . 134
10.2.1. Header . . . 134
10.2.2. Footer . . . 135
10.2.3. Menu Position . . . 136
10.2.4. Font Size . . . 136
10.2.5. Resource Files . . . 136
10.3. Customizing the Layout - Step by Step . . . 137
10.3.1. Header . . . 137
10.3.1.1. Standard Layout . . . 137
10.3.1.2. Hide Product Name . . . 137
10.3.1.3. Replace Company Name with Company Logo . . . 137
10.3.1.4. Hide Font Size Option . . . 137
10.3.1.5. Hide Company Name and Logo . . . 137
10.3.1.6. Hide Keyvisual and Show Logout Link . . . 138
10.3.1.7. Hide Options and Welcome Message . . . 138
10.3.1.8. Hide Top and Bottom Part of Header . . . 138
10.3.2. Footer . . . 138
10.3.2.1. Standard Layout . . . 138
10.3.2.2. Show Version and Suite . . . 138
10.3.2.3. Hide Ruler and Copyright . . . 138
10.3.2.4. Hide Entire Footer . . . 139
10.3.3. Menu . . . 139
10.3.3.1. Display Menu Vertically . . . 139
11. Web Center Menus . . . 140
11.1. Menu Overview . . . 140

11.1.1. Menu Access Policies . . . 140
11.1.2. Navigation Menu . . . 140
11.1.2.1. Menu Handling . . . 140
11.1.2.2. Menu Definition . . . 141
11.1.2.3. Menu Labels . . . 141
11.1.2.4. Navigation Bar Location . . . 142
11.1.2.5. Assigning a Menu to a Navigation Bar . . . 142
11.1.2.6. Generated Javascript Code . . . 143
11.1.2.7. Menu Display . . . 143
11.1.2.8. Requests on Menu Selection . . . 143
11.1.3. Context Menus . . . 144
11.1.3.1. Context Menu Handling . . . 144
11.1.3.2. Context Menu Definition . . . 145
11.1.3.3. Context Menu Assignments . . . 145
11.1.3.4. Generated Javascript Code . . . 146
11.1.3.5. Context Menu Display . . . 147
11.1.3.6. Requests on Context Menu Selection . . . 147
11.1.3.7. Context Menu Handler Actions . . . 147
12. Form and List Frames . . . 149
12.1. Overview . . . 149
12.2. Titles . . . 149
12.2.1. Defining Titles . . . 150
12.2.1.1. Sample . . . 150
12.2.2. Assigning Titles . . . 150
12.2.2.1. A form with title and subtitle . . . 150
12.3. Toolbars . . . 151
12.3.1. Defining Toolbars . . . 151
12.3.1.1. Form Toolbars . . . 151
12.3.1.1.1. Sample . . . 151
12.3.1.2. List Toolbars . . . 151
12.3.1.2.1. Sample . . . 152
12.3.2. Icons . . . 152
12.3.3. Labels and Tooltips . . . 152
12.3.3.1. Sample . . . 153
12.3.4. Assigning Toolbars . . . 153
12.3.4.1. Form Toolbars . . . 153
12.3.4.1.1. Sample . . . 153
12.3.4.2. List Toolbars . . . 153
12.3.4.2.1. Sample . . . 153
12.4. Page Size Selectors . . . 154
12.4.1. Defining Page Size Selectors . . . 154
12.4.1.1. Sample . . . 154

12.4.2. Assigning Page Size Selectors . . . 154
12.4.2.1. Sample . . . 155
12.5. Global Options . . . 155
12.5.1. Form Frames . . . 155
12.5.1.1. Sample . . . 156
12.5.2. List Frames . . . 156
12.5.2.1. Sample . . . 156
13. Language and Font Size Chooser . . . 158
13.1. Language Chooser . . . 158
13.1.1. Valid Languages . . . 158
13.1.2. Configuring the Language Chooser . . . 158
13.1.3. Defining the Default Language . . . 158
13.1.4. Removing the Chooser . . . 159
13.2. Font Size Chooser . . . 159
13.2.1. Valid Font Size Names . . . 159
13.2.2. Available Font Size Names . . . 159
13.2.3. Configuring the Font Size Chooser . . . 159
13.2.4. Defining the Default Font Size . . . 160
13.2.5. Removing the Chooser . . . 160
13.3. Removing Both Choosers . . . 160
13.4. Reset Points . . . 161
14. Utility Bar . . . 162
14.1. Utility Bar Definition . . . 162
14.2. Navigation History . . . 163
14.2.1. Maximum Number of Items . . . 163
14.2.2. Configuring Tooltips and Labels . . . 163
14.3. Quick Search . . . 164
14.3.1. Configuring Tooltips and Labels . . . 165
14.4. Advanced Search . . . 166
14.4.1. Configuring the Link Text . . . 166
14.5. Global Options . . . 166
15. Home Page . . . 168
15.1. Defining the Home Menu . . . 169
15.2. Displaying the Home Page after Login . . . 169
15.3. Defining the Home Action . . . 170
15.4. Defining the Home Page . . . 170
15.5. Property Plug-Ins . . . 171
15.5.1. Form Definition . . . 172
15.5.1.1. Sample . . . 172
15.5.2. Struts Action . . . 173
15.5.2.1. Sample . . . 173
15.5.3. Tiles Definition . . . 173

15.5.3.1. Sample ... 173
15.6. List Plug-Ins ... 174
15.6.1. Form Definition ... 174
15.6.1.1. Form Property List ... 174
15.6.1.2. List Property ... 175
15.6.1.3. Samples ... 176
15.6.1.3.1. Sample 1 ... 176
15.6.1.3.2. Sample 3 ... 176
15.6.2. Struts Action ... 177
15.6.2.1. Sample ... 178
15.6.3. Tiles Definition ... 178
15.6.3.1. Sample ... 178
15.7. Global Properties ... 179
16. Sorting Lists ... 180
16.1. Sort Attributes ... 180
16.1.1. List Attributes ... 180
16.1.2. Column Attributes ... 180
16.2. Sort Attribute Evaluation ... 180
16.3. Samples ... 180
17. Expression Language Extensions ... 183
17.1. Defining Functions ... 183
17.1.1. Attribute Exists ... 183
17.1.2. Attribute Contains ... 183
17.1.3. Attribute Matches ... 184
17.2. Assigning the Functions to a Scoped Variable ... 184
17.3. Using the Functions ... 185
17.3.1. Menu Selectors ... 185
17.3.2. JSP Pages ... 185
18. Accessibility ... 187
18.1. Labels ... 187
18.1.1. Attribute aria-label ... 187
18.1.2. Attribute aria-labelledby ... 188
18.1.3. Fieldset ... 188
18.2. ARIA Roles ... 189
18.2.1. Presentation ... 189
18.2.2. Menubar and Toolbar ... 189
18.3. Read-only Fields in Tab Order ... 189
18.4. Editable Date Fields ... 190
18.5. Table Summaries ... 191
19. Using the Keyboard ... 193
19.1. Fully Accessible User Interface Controls ... 193
19.1.1. Main Menu ... 193

19.1.1.1. If the focus is on a menu title: 193
19.1.1.2. If the focus is on a menu item: 193
19.1.1.3. Shortcuts 193
19.1.2. Form and List Toolbar 194
19.1.2.1. If the focus is on a tool: 194
19.2. Other User Interface Controls 194
19.2.1. Calendar Widget 194
19.2.1.1. If the focus is on a date: 194
19.2.2. Tree Widget 194
19.2.2.1. If an item is highlighted: 194
19.2.3. Edit Control for Multi-valued Properties 195
19.2.3.1. If the focus is on a value: 195
19.2.4. List 195
19.2.4.1. If the focus is on a list header cell: 195
19.2.4.2. If the focus is on a list body cell: 195
19.2.5. List Context Menu 196
19.2.5.1. If the focus is on a menu item: 196
20. Search Panel Configuration 197
20.1. Filter Attributes 197
20.1.1. Filtering by Attributes with a Fixed Value Set 198
20.1.1.1. Defining the Localized Values 198
20.1.1.2. Configuring the Search Filter Attribute 199
20.1.2. Filtering with Proposal Lists 199
20.1.2.1. Configuring the Search Filter Attribute 199
20.1.3. Filtering by Link Attributes 199
20.1.3.1. Configuring the Search Filter Attribute 200
20.1.3.2. Configuring the Search for the Linked Objects 201
21. Adding Languages 202
21.1. Overview 202
21.1.1. Files to Be Translated 202
21.1.2. How to Proceed 203
21.2. Translating Files 203
21.2.1. Message Files 203
21.2.1.1. File messages.js 204
21.2.1.2. File text.properties 204
21.2.2. Snippets 205
21.2.2.1. JSP Snippets 205
21.2.2.2. HTML Snippets 205
21.2.3. Error Pages 206
21.2.3.1. File noscript.html 206
21.2.3.2. File errorMessages.txt 206
21.2.3.3. File forbidden.html 207

21.2.3.4. File severeErrorPage.html . . . 207
21.2.4. Character Representations . . . 208
21.2.4.1. HTML Code . . . 208
21.2.4.1.1. Samples . . . 209
21.2.4.2. Java and JSON Unicode Character Representations . . . 209
21.2.4.2.1. Sample . . . 209
21.2.5. Translation Utilities . . . 209
21.2.5.1. text.properties . . . 210
21.2.5.2. messages.js . . . 210
21.3. Configuration Tasks . . . 210
21.3.1. Extending the Language Chooser . . . 210
21.3.2. Changing the Default Language . . . 211
21.3.3. How a User's Language is Determined . . . 211
22. Confirmations . . . 212
22.1. Overview . . . 212
22.2. Condition Check Functions . . . 212
22.2.1. Check Function Interface . . . 212
22.2.2. Predefined Checks . . . 213
22.2.3. Registering Checks . . . 213
22.3. Form Property Configuration . . . 214
23. Validation of Editable Input Fields in Tables . . . 215
23.1. Overview . . . 215
23.2. Samples . . . 215
23.2.1. Date Validations . . . 215
23.2.2. Time Validations . . . 216
23.3. Configuration . . . 216
23.3.1. Checks . . . 216
23.3.1.1. Sample . . . 217
23.3.2. Naming Columns . . . 217
23.3.2.1. Sample . . . 217
23.3.3. Validators . . . 217
23.3.3.1. Single-Property Validators . . . 217
23.3.3.2. Mutliple-Property Validators . . . 218
24. Customizing the Help Link in the Menu Bar . . . 220
24.1. Overview . . . 220
24.2. Customizing the Help Button in the Menu Definition . . . 220
24.2.1. Defining the Help Menu . . . 220
24.2.2. Adding the Help Menu to a Menu Bar . . . 221
24.3. Providing the Help Files . . . 221
24.4. Customizing the Help Window . . . 221
24.4.1. General Properties . . . 221
24.4.2. Per-Language Properties . . . 222

25. Rendering Attribute Values via Icons . . . 223
25.1. Overview . . . 223
25.2. Attribute Risk Level . . . 223
25.3. Displaying the Risk Level in Web Center . . . 223
25.4. Configuring Web Center . . . 224
25.4.1. Defining the Icon Renderers . . . 224
25.4.1.1. The Base Icon Renderer: . . . 224
25.4.1.2. The Risk Level Renderer: . . . 225
25.4.2. Setting the Default Renderer . . . 226
25.4.3. Adjusting the Icon Layout via CSS . . . 226
25.4.4. Providing the Icons . . . 226
25.4.5. Configuring Forms . . . 227
25.4.5.1. Form Property . . . 227
25.4.5.2. Data Property . . . 227
25.4.6. Localizing Labels and Tooltips . . . 227
25.4.6.1. Icon Tooltips . . . 228
25.4.6.2. Labels . . . 228
26. Enabling User Self-Registration . . . 229
26.1. Overview . . . 229
26.2. Enable Self-Registration Section on Login Page . . . 229
26.3. Activate the Self-Registration Actions . . . 229
26.4. Set the Self-Registration Password . . . 229
Legal Remarks . . . 231

Preface

The *DirX Identity Web Center Customization Guide* is designed to help you to customize the DirX Identity Web Center. It consists of the following sections:

- Chapter 1 provides information about the requirements to run Web Center and describes how to prepare your development environment.
- Chapter 2 describes how to separate customizations from the product files in order to ease upgrades to later Web Center versions.
- Chapter 3 describes several examples of how to customize Web Center.
- Chapter 4 provides information about renderers.
- Chapter 5 describes how to validate form input on the client side.
- Chapter 6 describes how to view photos and certificates.
- Chapter 7 describes how to configure the representation of distinguished names (DNs) within Web Center.
- Chapter 8 describes how to configure attribute modification approval within Web Center.
- Chapter 9 describes how to localize the values of a property with a finite number of values.
- Chapter 10 describes how to configure the basic layout of Web Center pages.
- Chapter 11 provides information about Web Center menus.
- Chapter 12 describes how to customize form and list frames.
- Chapter 13 describes how to customize language and font size chooser.
- Chapter 14 describes the utility bar with the quick search panel and the advanced search link.
- Chapter 15 describes the components of a home page.
- Chapter 16 describes how to configure the way a list is sorted.
- Chapter 17 describes expression language extensions.
- Chapter 18 handles some accessibility topics.
- Chapter 19 lists the keys supported by the Web Center user interface controls.
- Chapter 20 describes how to configure search panels.
- Chapter 21 provides details about how to add support for new languages to a Web Center application.
- Chapter 22 describes how form submit confirmations work.
- Chapter 23 describes how to validate editable fields in tables on the client side.
- Chapter 24 describes how to customize the link to an external help file in the menu bar.
- Chapter 25 describes how to render attribute values via icons.
- Chapter 26 describes how to enable user self-registration.

DirX Identity Documentation Set

Version: 9.0.2 | Date: 2026-11-02 | Timestamp: 13:57:36 UTC

The DirX Identity document set consists of the following manuals:

- *DirX Identity Introduction*. Use this book to obtain a description of DirX Identity architecture and components.
- *DirX Identity Release Notes*. Use this book to understand the features and limitations of the current release. This document is shipped with the DirX Identity installation as the file **release-notes.pdf**.
- *DirX Identity History of Changes*. Use this book to understand the features of previous releases. This document is shipped with the DirX Identity installation as the file **history-of-changes.pdf**.
- *DirX Identity Tutorial*. Use this book to get familiar quickly with your DirX Identity installation.
- *DirX Identity Provisioning Administration Guide*. Use this book to obtain a description of DirX Identity provisioning architecture and components and to understand the basic tasks of DirX Identity provisioning administration using DirX Identity Manager.
- *DirX Identity Connectivity Administration Guide*. Use this book to obtain a description of DirX Identity connectivity architecture and components and to understand the basic tasks of DirX Identity connectivity administration using DirX Identity Manager.
- *DirX Identity User Interfaces Guide*. Use this book to obtain a description of the user interfaces provided with DirX Identity.
- *DirX Identity Application Development Guide*. Use this book to obtain information how to extend DirX Identity and to use the default applications.
- *DirX Identity Customization Guide*. Use this book to customize your DirX Identity environment.
- *DirX Identity Integration Framework*. Use this book to understand the DirX Identity framework and to obtain a description how to extend DirX Identity.
- *DirX Identity Web Center Reference*. Use this book to obtain reference information about the DirX Identity Web Center.
- *DirX Identity Web Center Customization Guide*. Use this book to obtain information how to customize the DirX Identity Web Center.
- *DirX Identity Resolution Service Setup Guide*. Use this book to set up the DirX Identity Resolution Service as a standalone service.
- *DirX Identity Meta Controller Reference*. Use this book to obtain reference information about the DirX Identity meta controller and its associated command-line programs and files.
- *DirX Identity Connectivity Reference*. Use this book to obtain reference information about the DirX Identity agent programs, scripts, and files.
- *DirX Identity Troubleshooting Guide*. Use this book to track down and solve problems in your DirX Identity installation.

- *DirX Identity Installation Guide*. Use this book to install DirX Identity.
- *DirX Identity Migration Guide*. Use this book to migrate from previous versions.
- *DirX Identity REST Service Guide*. Use this book to migrate from previous versions.

Notation Conventions

Boldface type
In command syntax, bold words and characters represent commands or keywords that must be entered exactly as shown.

In examples, bold words and characters represent user input.

Italic type
In command syntax, italic words and characters represent placeholders for information that you must supply.

[ ]
In command syntax, square braces enclose optional items.

{ }
In command syntax, braces enclose a list from which you must choose one item.

In Tcl syntax, you must actually type in the braces, which will appear in boldface type.

|
In command syntax, the vertical bar separates items in a list of choices.

...
In command syntax, ellipses indicate that the previous item can be repeated.

userID_home_directory
The exact name of the home directory. The default home directory is the home directory of the specified UNIX user, who is logged in on UNIX systems. In this manual, the home pathname is represented by the notation *userID_home_directory*.

install_path
The exact name of the root of the directory where DirX Identity programs and files are installed. The default installation directory is *userID_home_directory*/**DirX Identity** on UNIX systems and **C:\Program Files\DirX\Identity** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathnames is represented by the notation *install_path*.

dirx_install_path
The exact name of the root of the directory where DirX programs and files are installed. The default installation directory is *userID_home_directory*/**DirX** on UNIX systems and **C:\Program Files\DirX** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathname is represented by the notation *dirx_install_path*.

dxi_java_home
The exact name of the root directory of the Java environment for DirX Identity. This location is specified while installing the product. For details see the sections "Installation" and "The Java for DirX Identity".

tmp_path

The exact name of the tmp directory. The default tmp directory is /tmp on UNIX systems. In this manual, the tmp pathname is represented by the notation *tmp_path*.

tomcat_install_path
The exact name of the root of the directory where Apache Tomcat programs and files are installed. This location is defined during product installation.

mount_point
The mount point for DVD device (for example, **/cdrom/cdrom0**).

1. Getting Started

This chapter provides information about the requirements for running DirX Identity Web Center and how to prepare your development environment.

1.1. Deploying Web Center

This section describes how to deploy Web Center using the DirX Identity Configuration program and how to deploy multiple Web Center instances manually.

1.1.1. Deploying Web Center with the DirX Identity Configuration Program

Web Center is usually deployed into Tomcat with the DirX Identity Configuration program. Since Web Center requires Tomcat 11 running with Java 21, make sure that an appropriate Tomcat is installed and runs with Java 21 before starting the Web Center configuration.

After you install and configure DirX Identity, you will find the Web Center application in the folder *install_path***/web/instances/webCenter-***domain***/webCenter**.

The context descriptor file **webCenter-***domain***.xml** serves to integrate the Web Center application into Tomcat.The file must be located in Tomcat's context descriptor folder.The exact name of the folder depends on Tomcat configuration details.The general name scheme is *tomcat_home***/conf/***enginename*/*hostname*.On most servers, the folder is named *tomcat_home***/conf/Catalina/localhost** or *tomcat_home***/conf/Standalone/localhost**.

The context descriptor file contains a Context element with a docBase attribute pointing to the Web Center installation directory, for example

```
<Context docBase="C:/Program
Files/DirX/Identity/web/instances/webCenter-My-Company/webCenter" …>
```

The DirX Identity Configuration creates the file and fills in the correct docBase. The docBase points to the folder *install_path***/web/instances/webCenter-*domain*/webCenter**.

The configuration program changes some settings in **web.xml** (like LDAP server address and domain name) and **password.properties**.

The DirX Identity Configuration also copies the jar file **dxmStorageURL.jar** to folder *tomcat_home***/dxilib**. The folder is created if not yet existing. The jar file is added to Tomcat's Java classpath.

On Windows systems where Tomcat is installed as a Windows service the configuration program stops and restarts the service.

After configuration, check that the context descriptor file has been created and refers to the correct document base. Check the content of **web.xml** and **password.properties**. Make

sure that the **dxilib** folder exists and contains the required library and that Tomcat's Java classpath has been updated. And finally check that Tomcat is up and running. If not, start it.

To start Web Center, just open a browser (supported are reasonably new versions of Edge, Internet Explorer, Firefox and Chrome) and enter the URL

`http://localhost:8080/webCenter-domain`



Host name, port and scheme (http or https) may vary depending on Tomcat configuration. The application path (webCenter-*domain*) is identical to the name of the context descriptor file. If you rename the file to **myWebCenter.xml** and use HTTPS on port 8443, the URL changes to

`https://localhost:8443/myWebCenter`

Deployment of Web Center via the DirX Identity Configuration may fail (in parts or totally) for example due to missing permissions to create or modify files in the Tomcat installation folder or insufficient access rights to stop and start the Windows service.

1.1.2. Deploying Web Center Manually

You can manually deploy another Web Center instance

- Into the same Tomcat server the default instance is deployed into.
- Into a different Tomcat server on the same machine.
- Into a Tomcat server on another machine. You don't even need to install DirX Identity on this machine.

To deploy a Web Center instance manually:

- Stop Tomcat.
- On the machine with the Tomcat server, create a new Web Center application directory *webCenter_app_path* (Don't create it in Tomcat's **webapps** folder).
- Copy the application folder **webCenter** from the folder *install_path***/web/instances/webCenter-***domain* to *webCenter_app_path*. Note that *install_path* refers to the DirX Identity installation folder, which may reside on a different machine.
- Copy the context descriptor file **webCenter.xml** from folder *install_path***/web/webCenter.template** to Tomcat's context descriptor folder; for example, *tomcat_home***/conf/Catalina/localhost**.
- Edit file **webCenter.xml**. Fill in the correct value for the docBase attribute of the Context element:

  ```
  <Context docBase="webCenter_app_path/webCenter" …>
  ```

- Navigate to the directory *webCenter_app_path***/webCenter/WEB-INF**.
- In file **web.xml**, configure the LDAP server connection and the DirX Identity domain.

- In file **password.properties**, enter the password for the domainAdmin user of your Provisioning domain with the key **ldap**.
- Create the folder *tomcat_home***/dxilib** if it doesn't exist already.
- Copy the following file to folder *tomcat_home***/dxilib**:
  - *install_path***/webCenter.template/endorsed/lib/dxmStorageURL.jar**
- Add the jar file to Tomcat's Java classpath.
- Make sure that the Tomcat server has the access rights to load the libraries in the **dxilib** folder and in Web Center's **WEB-INF/lib** folder and to read the files of the Web Center application and the context descriptor file.
- Start Tomcat.

Now you can access your Web Center instance with the URL

`http://localhost:8080/webCenter`

1.1.2.1. Deploying Additional Web Center Instances

You can deploy additional Web Center instances into the same Tomcat service by just copying the context descriptor **webCenter.xml** to a new context path; for example, **webCenterCust.xml**.

If you don't change the docBase in the new file, both instances will use the same application folder so that changes to the files in the application folder affect both instances.The only good reason to deploy two applications this way is to assign different values to the configuration parameters (like Struts configuration file names) in the new deployment descriptor.

You may as well copy the application folder of the first instance and let the docBase in the new instance's context descriptor point to the new folder.This way, changes to one application folder don't affect the other application.

Note that Tomcat monitors the context descriptor folder.On detection of a new context descriptor, Tomcat loads the new application.On detection of a changed context descriptor (new modification date), Tomcat reloads the affected application.

Therefore, Tomcat will automatically load the new Web Center instance after a short time, and you can access it via the URL

`http://localhost:8080/webCenterCust`

1.2. Preparing the Development Environment

To set up an environment for Web Center development, we recommend that you start Tomcat as a console application rather than running it as a service.The advantages of running Tomcat as console application are:

- You can start Tomcat faster by double-clicking the Tomcat icon on the desktop.

- You can stop Tomcat faster by closing the Tomcat window.
- You can view debug messages in the Tomcat console window.
- You can view error messages caused by syntax errors in the configuration files.

To run Tomcat as a console application:

- Stop the Tomcat service.
- Change the startup type of the service to manual.
- Navigate to folder *tomcat_home***/bin**.
- Create a shortcut for **tomcat11.exe**.
- Place the shortcut on the desktop.

Next, create a copy of your Web Center default installation as described in the section “Deploying Web Center". Using a copy of the Web Center installation allows you to configure Web Center without changing the original product.

In the file **WEB-INF/web.xml**, enable debug mode by setting the context parameter **com.siemens.webMgr.log.level** to level 2:

```
<context-param>
    <!-- Allowed debug switches: 0=SEVERE,1=INFO,2=FINEST,
                                 3=WARNING,-2=log4j.properties -->
    <param-name>com.siemens.webMgr.log.level</param-name>
    <param-value>2</param-value>
</context-param>
```

2. Customization Files

2.1. Properties Files

2.1.1. Web Center Properties

The Web Center properties file

> **WEB-INF/config/webCenter.properties**

contains configuration properties to adapt the overall layout of Web Center pages and to adjust some features. To override a default property value, add a property with identical name and the new value to file

> **WEB-INF/config/webCenterCustom.properties.**

You can also add new properties to the custom file. The custom file properties take precedence over the default ones.

The properties are made available to the application as an application-scoped variable of type java.util.Map with name “webCenter”. In a JSP, for example, the expression “${webCenter.initialSearch}” is resolved to the value of the property with name “initialSearch”. Its default value is “false”. In order to change it to “true”, add the following line to the custom properties file:

> **initialSearch = true**

The custom property file is not delivered with Web Center, so create it if needed.

2.1.2. Paths

The path configuration file

> **WEB-INF/config/paths.properties**

contains a couple of action and JSP lists. The lists were defined in checkSession.jsp in previous versions but have been moved to a separate configuration file for easier adaptation.

To override a default path, add a property with identical name and the new value to file

> **WEB-INF/config/pathsCustom.properties.**

The custom paths override the default ones.

2.2. Message Texts

The default message texts are defined in

> **WEB-INF/classes/resources/languages/<language>/text.properties.**

In order to customize a default message, add a message with the same key and the customized text to file

> **WEB-INF/classes/resources/languages/<language>/text_<language>.properties**

for each language your application supports.

For example, the custom message file for the English language is

> **WEB-INF/classes/resources/languages/en/text_en.properties.**

You can, of course, also add additional messages to the custom files.

The custom message files are not delivered with Web Center, so create them if needed.

Customization Sample

Default Message File: text.properties

```
application.logonStatus.loggedOn = Logged on as
application.shortTitle           = User Management with DirX
```

Custom Message File: text_en.properties

```
application.logonStatus.loggedOn = You are logged on as
application.shortTitle           =
custom.companyName               = Your-Company
```

The custom file

- Redefines the text of the 1st message.
- Removes the text of the 2nd message, which means no short title will be displayed.
- Defines a new custom message with key “custom.companyName”.

2.3. Objects Configuration

The standard Web Center objects and object relations are configured in file

> **WEB-INF/config/objects-config.xml.**

The location of the file can be changed in the context descriptor file

> **webCenter-<domain>.xml:**
>
> ```
> <Parameter override="false" name="ObjectConfig.Path"
>           value="/WEB-INF/config/objects-config.xml"/>
> ```

Do not change the standard definition file or location. Instead, place your custom object definitions in a new file and append its path name to **ObjectConfig.Path**, for example

```
<Parameter override="false" name="ObjectConfig.Path"
          value="/WEB-INF/config/objects-config.xml,
                 /WEB-INF/custom/config/objects-config.xml"/>
```



A custom definition will overwrite any default definition with the same object id or relation name.

Customization Sample

Default File: WEB-INF/config/objects-config.xml

```
<objects>
       …
       <object id="project"
               nodeClass="siemens.dxr.organization.nodes.SvcProject"
               set="Projects"
               dirClass="dxrProject"/>
       …
</objects>
```

Custom File: WEB-INF/custom/config/objects-config.xml

```
<objects>
      <object id="project"
```

```
              nodeClass="custom.customObject"
              set="Projects"
              dirClass="dxrProject"/>
</objects>
<relations>
      <relation name="customAssigned" value=".assigned"/>
<relations>
```

The custom file

- Redefines the default definition of object “project”.
- Defines a new relation with id “customAssigned”.

2.4. Renderers

2.4.1. Renderer Definitions

The definitions of the standard Web Center renderers are configured in file

> **WEB-INF/config/renderers-config.xml.**

The location of the file can be changed in the context descriptor file

> **webCenter-<domain>.xml:**
>
> ```
> <Parameter override="false" name="RendererConfig.Path"
>           value="/WEB-INF/config/renderers-config.xml"/>
> ```

Do not change the standard definition file or location. Instead, place your custom renderers in a new file and append its path name to **RendererConfig.Path**, for example

```
<Parameter override="false" name="RendererConfig.Path"
          value="/WEB-INF/config/renderers-config.xml,
                 /WEB-INF/custom/config/renderers-config.xml"/>
```



A custom renderer will overwrite any default renderer with the same id.

Customization Sample

Default File: WEB-INF/config/renderers-config.xml

```
<renderer id="text"
      defURL="/WEB-INF/snippets/textField/roTextField.htm"
      altURL="/WEB-INF/snippets/textField/rwTextField.htm"
      jsURL="/WEB-INF/snippets/textField/textField.js"
      roJsURL="/WEB-INF/snippets/textField/roTextField.js/>
```

Custom File: WEB-INF/custom/config/renderers-config.xml

```
<renderer id="text"
      defURL="/WEB-INF/custom/snippets/textField/roTextField.htm"
      altURL="/WEB-INF/custom/snippets/textField/rwTextField.htm"
      jsURL="/WEB-INF/snippets/textField/textField.js"
      roJsURL="/WEB-INF/snippets/textField/roTextField.js/>
<renderer id="customText" extends="text"
      roJsURL="/WEB-INF/snippets/textField/roTextField.js/>
```

The custom file

- Redefines the default definition of renderer “text”.
- Defines a new renderer with id “customText”.

2.4.2. Default Renderer Assignments

Default assignments of renderers to form properties are configured in file:

> **WEB-INF/config/defaultRenderer.properties.**

The location of the file is configured in the properties file

> **WEB-INF/config/webCenter.properties**
>
> ```
> config.defaultRenderers = /WEB-
> INF/config/defaultRenderer.properties
> ```

Do not change the standard definition file or location. Instead, place your custom assignments in a new file and append its path name to **config.defaultRenderers** in file

> **WEB-INF/config/webCenterCustom.properties**

For example:

```
config.defaultRenderers =
       /WEB-INF/config/defaultRenderer.properties,
       /WEB-INF/custom/config/defaultRenderer.properties
```



A custom assignment will overwrite any default assignment with the same key.

Customization Sample

Default File: WEB-INF/config/defaultRenderer.properties

```
java.lang.String = text
manager          = userSearch
```

Custom File: WEB-INF/custom/config/defaultRenderer.properties

```
java.lang.String = customText
custommember     = userLinks
```

The custom file

- Redefines the default renderer for properties of type “java.lang.String”.
- Adds a new assignment for property “customMember”.

2.5. Including Custom Javascript Files and Stylesheets

The main Web Center layout page adds the content of a custom JSP to the header of each HTML page. The JSP is intended to contain <script> and <style> tags to include custom Javascript files and stylesheets for the page.

The name of the file must be configured as Web Center property with name “customScriptsAndStyles”, e.g.

```
customScriptsAndStyles = /WEB-INF/custom/jsp/ScriptsAndStyles.jsp
```

Specify the file name as an absolute path from the Web Center root folder. We recommend putting the file somewhere below the **WEB-INF** folder since this prevents the file from being downloaded to clients.

The following sample file includes a script and a stylesheet:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<script src="resources/custom/scripts/custom.js"></script>
<script src="resources/custom/messages/
       <c:out
value="${sessionScope['com.siemens.webMgr.language']}"/>/
       messages.js">
</script>
<link rel="StyleSheet" type="text/css"
      href="resources/custom/styles/styles.css">
```



Including files in an HTML page has an impact on performance. It's not advisable to include too many files.

2.6. Javascript

2.6.1. Custom Code

Do not change any of the Javascript files supplied with Web Center in order to add custom script code. Keep all your code in a separate custom file.

2.6.2. Custom Renderers

Custom renderers for table cells must be registered with the renderers object defined in Javascript file **documentFuncs.js**.

As of DirX Identity 8.2, the renderers object provides some methods to register custom renderers. So there is no more need to change the file **documentFuncs.js**.

For details on custom renderers, see chapter "Renderers."

2.6.3. Custom Validators

Custom validators must be registered with the validators object defined in Javascript file **validation.js**.

The validators object provides a method to register custom validators. So there is no need to change the file **validation.js**.

For details on custom validators, see chapter "Validation."

2.6.4. Custom Messages

Each Web Center page loads a language dependent Javascript message file **resources/build/messages/<language>/messages.js.**

The default messages are defined in object messages.texts. At the end of each file, the

object messages.custom.texts is defined.

Now, in order to overwrite a default message text messages.texts, create your own language-specific message file and add a property with the name to messages.custom.texts. You can also add additional messages to the messages.custom.texts.

Then include the custom message file in each Web Center page as described in section “Including Custom Javascript Files and Stylesheets” above.

2.6.4.1. Customization Sample

2.6.4.1.1. Default Message File

```
messages.texts = {
    ...
    attributes: {
        cn:              "Name",
        description:     "Description",
        $displayname:    "Name",
        dxmactive:       "Active",
        ou:              "Department",
        telephonenumber: "Phone"
    }
};
```

2.6.4.1.2. Custom Message File

```
messages.custom.texts = {
    ...
    attributes: {
        cn:              "Common Name",
        $displayname:    "Displayname",
        ou:              "Organizational Unit",
        telephonenumber: "Phone Number"
    },
    colors: {
        red:   "Red",
        blue:  "Blue",
        green: "Green"
    }
};
```

The custom file overrides some of the default texts in the “attributes” section, and defines a new section “colors”.

When editing a message file take care of correct syntax, esp. of properly separating two adjacent definitions with a comma. There mustn’t be a comma after the last definition with each section, e.g. at the end of the “telephonenumber” line, or after the closing curly brace of the “colors” section in the above sample.

2.7. Custom File Overview

Custom files might be located below an application’s root folder in the following way:

```
resources
    custom
        messages
            de
                messages.js
            en
                messages.js
        scripts
            custom.js
        styles
            small
                styles.css
            medium
                styles.css
            large
                styles.css
WEB-INF
    classes
        resources
            languages
                de
                    text_de.properties
                en
                    text_en.properties
    config
        webCenterCustom.properties
    custom
        config
            converters.properties
            defaultRenderers.properties
```

```
            messages.properties
            objects-config.xml
            renderers-config.xml
        jsp
            scriptsAndStyles.jsp
```

You may choose different folders and names for your custom files. Files with fixed names and locations are printed in bold font.

2.8. Configuring Customization File Names

The following table summarizes where to configure the names of customization files.

Configuration Files with Fixed Names

File Name	Customization File
WEB-INF/config/webCenter.properties	WEB-INF/config/ webCenterCustom.properties
WEB-INF/classes/resources/ languages/<language>/text.properties	WEB-INF/classes/resources/ languages/ <language>/ text_<language>.properties

Configuration Files with Variable Names

File Name	Configuration File	Parameter
WEB-INF/config/../ struts-config.xml	webCenter-<domain>.xml	Struts.DefaultFiles Struts.Path
WEB-INF/config/../ forms-config.xml	webCenter-<domain>.xml	Struts.DefaultFiles Struts.Path
WEB-INF/config/../ tiles-defs.xml	webCenter-<domain>.xml	Struts.DefaultFiles Struts.Path
WEB-INF/config/../ menu-defs.xml	webCenter-<domain>.xml	Struts.DefaultFiles Struts.Path
WEB-INF/config/ converters.properties	WEB-INF/config/ webCenterCustom.properties	config.converters

File Name	Configuration File	Parameter
WEB-INF/config/ defaultRenderers.properties	WEB-INF/config/ webCenterCustom.properties	config.defaultRenderers
WEB-INF/config/ messages.properties	WEB-INF/config/ webCenterCustom.properties	config.messages
WEB-INF/config/ objects-config.xml	webCenter-<domain>.xml	ObjectConfig.Path
WEB-INF/config/ renderers-config.xml	webCenter-<domain>.xml	RendererConfig.Path
WEB-INF/config/ validation.xml	WEB-INF/config/identity/ struts-config.xml	ValidatorPlugIn: pathnames
WEB-INF/config/ validator-rules.xml	WEB-INF/config/identity/ struts-config.xml	ValidatorPlugIn: pathnames
Custom Javascript files	Custom include file	script tags
Custom CSS stylesheets	Custom include file	link tags
Custom include file	WEB-INF/config/ webCenterCustom.properties	customScriptsAndStyles

The table lists the configuration files for the WebCenter application.

The corresponding files for the Self-Service application are **selfService-*domain*.xml** and **WEB-INF/config/identitySelf/struts-config.xml**.

3. Customization Examples

The following sections describe several examples of how to customize Web Center. These examples range from very simple changes to Web Center layout to adding your own pages with new functionality.

3.1. Changing the JSP Page Layout

If the Web Center layout delivered with DirX Identity does not match your requirements, you can change the layout of the JSP pages.

The JSP pages for resulting HTML pages (that is, the view) are located in the **WEB-INF/jsp/view** directory of the installation.This directory contains the directories **forms** and **tiles**. The **forms** define complete pages, while **tiles** contain the "building blocks" for pages.

3.1.1. Common Layout Template layout.jsp

All JSP pages used by Web Center have a common basic tiles definition that is used to construct a final page with a set of page tiles. The item names used in the definition tag correspond directly with the layout template stored in the **layout.jsp** page that is located in the **jsp/view/forms** directory.

Inserting selected components from the **jsp/view/tiles** directory filled with dynamic content completes the view. To complete the page, the renderer implementation uses HTML and JavaScript code snippets from the **WEB-INF/snippets** folder, JavaScript files from the resources/build/scripts folder, and a cascading style sheet file stored in the **resources/build/styles** directory. Any graphics object is taken from the **resources/images** directory.

The layout chosen for the template conforms to the Fibonacci grid. The heights and widths, fonts, foreground and background colors of particular elements are defined in **styles.css**. Web Center delivers three different sets of cascading style sheets: small, medium and large. The user can select one in the sub-header part of each page.



The last tag inside the body tag of **layout.jsp** is <view:alert/>. This tag should not be removed, because it provides a message pop-up window when errors or notifications occur.

Changing the basic layout template can be a time-consuming task, since the entire body is an HTML table with partial percentage heights and widths. Removing an item may cause the page to be displayed incorrectly. Use an HTML analog and try the changes in an HTML editor before making any layout template modification.

The following sections provide a brief description of some of the styles as they are used in the Web Center configuration delivered with DirX Identity. You can add other styles for the individual design of controls. For the definition of style elements, see the books, articles and Web sites dealing with HTML on the Internet.

3.1.2. CSS Styles

Some styles in the style sheets differ between Internet Explorer and Firefox, some between Firefox 3 and later versions of Firefox.

The body element of each Web Center page has the HTML id “msie” in Internet Explorer, and “firefox” in other browsers (like Firefox and Chrome).

You can use this to define browser-dependent styles as shown in the following samples:

- Style “#msie .labelCell“ applies to Internet Explorer.
- Style “#firefox .labelCell” applies to all other browsers (like Firefox and Chrome).

3.1.2.1. Form Styles (styles.css)

.accountList	Form style for the account list (width).
.buttonGap	Space between last form field row and button row (padding).
.calWeekdayPanel	Layout for the calendar weekday panel (background-color, color, padding, font).
.calWindow	Basic layout of the calendar window (background-color and size).
.formLogin	Login form layout (width).
.imageButton	Image button layout (color, border, size). Used for the calendar button.
.labeledButton	Standard labeled button layout (background, border, color, cursor, font).
.pressedButton	Layout for a permanently pressed day button in the calendar panel (border, color, cursor, font).
.releasedButton	Layout for a released day button in the calendar panel (border, color, cursor, font).
.roLabel	Label layout for read-only text fields (border, color, font).
.rwLabel	Label layout for read-write text fields (border, color, font).
.roStringList	Layout for read-only string lists (background-color, border, color, font).
.rwStringList	Layout for read-write string lists (background-color, border, color, font, overflow, scrollbar).
.roTextField	Layout for read-only text fields (background-color, border, color, font).
.rwTextField	Layout for read-write text fields (background-color, border, color, font).

.roTokenString	Layout for read-only string lists (background-color, border, color, font, overflow).
.rwTokenString	Layout for read-write string lists (background-color, border, font, overflow).
.searchPanel	Layout for the search panel; that is, the panel containing the search controls (width).
.selfAssignList	Layout for the self-assignment list of roles or permissions (width).
.tableBody	Layout for the table body; that is, the area where all table rows are contained (background-color, overflow, scrollbar).
.tableBorder	Table border layout (border).
.tableHeaderCell	Layout for the header cells of a table (background-color, border-left, color, cursor, padding).
.tableInfo	Layout for the information field below a table (background-color, border, color, font).
.tableNavigationButton	Layout for the scrollbar buttons of a simple table (border, background-color, color, cursor, width).
.tableSelectedCell	Layout for the cell of a selected row of a table (background-color, border, color, cursor, padding).
.tableUnselectedCell	Layout for the cell of an unselected row of a table (background-color, border, color, cursor, padding).
.tabSelected	Tab layout for the selected tab button in a tab sheet (background).
.tab	Tab layout for unselected tab buttons in a tab sheet (background).
.tabDisabled	Tab layout for disabled tab buttons in a tab sheet (background, color)
.userList	User list form layout (width).

3.1.2.2. Page Styles (styles.css)

p,h1,h2,h3,h4,ul, ol,li,div,td,th, address,blockquote, nobr,b,i	Predefined text styles.
.bcnFont	Font style for the bread-crumb-navigation text.
.bodyText	Style of the text used in the text-content tile.
.footerFont	Font style used for the text in the footer.
.footerLine	Layout for the footer’s horizontal line (padding, width).

	Layout for the header area (background, padding, height).
.headlineText	Header font for the text-content.
.headerKeyVisual	Layout for the key visual in the upper-left corner of the page (size).
.leftContent	Style for the left content area (color and width).
.headerCompanyLogo	Layout for the company logo appearing in the header (height, padding).
.headerCompanyName	Layout for the company name appearing in the header (height, font, height).
	Layout for the horizontal and vertical secondary-navigation bar, respectively (color and width).
a:link,a:visited,a:active	Style for anchors.

3.2. Adding Languages

DirX Identity Web Center is delivered with the languages **en** (English) and **de** (German). You can add additional languages to this set.

A language variant is stored in a separate directory identified by the locale name (**en**, **de**, **fr**, **it**, and so on).The language directory is found in the directory **WEB-INF/classes/resources/languages**.

3.2.1. Content of a Language Directory

A language directory contains a set of **.html** and **.jsp** files and a **text.properties** file.T he **.html** and **.jsp** files contain the pure informal text for the respective pages (which may be in most cases derived from the name), while the **text.properties** file contains all texts used in the forms and in common parts of all pages.

3.2.2. Using Language-Dependent Texts

Language-dependent texts of labels and messages are contained in the **text.properties** file. The file contains the text fragments as key/value pairs; the key represents the message identifier and the value represents the message text.

These texts can be accessed in the JSP pages in various ways:

Java Standard Tag Library (JSTL):

- Using the JSTL tag <fmt:message>. You must include the **fmt** library into the JSP page as follows:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
```

Once you include this library, you can access any **text.properties** item to be used in a JSP page with the definition:

```
<fmt:message key="textKey"/>
```

where *textKey* is a valid key contained in the **text.properties** file and the tag is placed at exactly the position at which the text is required, since the tag is simply replaced with the key value. Use these keys also as error messages when developing controller JSP pages for forms and tiles configurations.

Web Center controller Library:

- Using the tag `<ctrl:setMessage>` of Web Center controller tag library. The tag stores the message key and optional message parameters in session-scoped variables. The variables are evaluated later on by the `<view:alert>` tag that generates a Javascript message to be displayed when the response page is loaded into the browser. Include the tag into your page as follows:

  ```
  ++<++%@ taglib
  uri="http://www.siemens.com/directory/webManager/controller"
  prefix="ctrl" %++>++
  ```

Then you can include the text that corresponds to the given key with the following definition:

```
<ctrl:setMessage key="application.error.missingSurname"/>
```

- A note about tag parameters:

Some tag implementations need to display labels or other text messages. Often they expect the message key as a parameter rather than the text fragment itself. See the respective tag library definitions in the *DirX Identity Web Center Reference*.

3.2.3. Using Text Files

The text files are included into a result page by tiles configuration. They are defined in the page attribute **textFile**:

```
<definition name=".typicalPage" extends=".base">
    <put name="bcn" value="login.title:/logout.do"/>
    <put name="formContent" value=".typicalPageForm"/>
    <put name="textFile" value="typicalPage.html"/>
</definition>
```

3.2.4. Adding a New Language

If you start with the English language, it is quite easy to add a new language to the Web Center by simply translating the key values in the **text.properties** file and the information texts contained in the **.html** and **.jsp** files.Do not touch the contained key names, because this can lead to erroneous output.

Store the translated files in a directory at the same level as the other language directories and give it the name of your locale.

For more detailed instructions refer to chapter “Adding Languages”.

3.3. Changing Attribute Lists in Pages

If you are satisfied with the default presentation and just want to display and modify some other attributes, you can simply change the form bean definitions.

Form beans are defined in files **forms-config.xml** as <form-bean> elements. A <form-bean> lists the attributes to be handled for the object in <form-property> child elements. For a property, you define its name, type, label and some other attributes as shown in the following excerpt for the property “c(ountry)”:

```
<form-property name="c"
               type="java.lang.String"
               label="ldap.attribute.c"
               readonly="true"
               width="100%" y="3" x="0"/>
```

Add and remove properties in the form bean definition as necessary. Make sure the properties exist in the DirX Identity domain with this name and type and are known to the respective object description.

Note that the label definition in the above sample can be omitted since the default label for a form property is “`ldap.attribute.`*`form property name`*”.

3.4. Removing Parts of the Default Application

Web Center ships with two application variants: A complete application that covers all management tasks and a self-service application. But what should you do if you just want to deploy part of the functionality?This section shows you how to perform this task simply by changing the configuration. For example, suppose we want to offer only role management in our application and strip out all the other functionality. This example also shows how to cut off part of the application.

3.4.1. Customizing the Context Configuration webCenter-*domain*.xml

First, deploy your new application in a separate folder. Copy the files from the default

application to a new folder, for example, to a folder *install_path*/**web/instances/rolemanagement**.

Next, provide a proper context configuration file and remove all Struts and form configuration files that you do not need.

In a Tomcat deployment, the default context configuration **webCenter-*domain*.xml** is located in the directory **conf/Catalina/localhost**. To create a separate application that is different from the delivered default, you can rename the context configuration, for example, to rolemanagement.xml. This action determines the context path for your new application: `<http://localhost:8080/rolemanagement>`. Change the context parameter **docPath** to point to the directory where you have copied your application files.

The parameter "Struts.Path" lists the folders containing the Struts and form bean configuration files for all your application modules. Keep only the folders you need. In our sample, these are **/WEB-INF/config/imports, /WEB-INF/config/identity**, **/WEB-INF/config/privileges** and **/WEB-INF/config/privileges/roles**.

3.4.2. Struts and Forms Configuration

Make sure that your application presents the proper start page at login. To accomplish this task, you need to change the settings in the Struts configuration file **identity/struts-config.xml**.

The action with which you want to start when the user has authenticated is defined as start action in file **identity/menu-defs.xml**. Change it to the search page for roles:

```
<definition name=".startActions">
    <put name="items" value=".menuRoleManagement:select"/>
</definition>
```

Then change the action "getStartAction" in **identity/struts-config.xml** accordingly:

```
<action path="/getStartAction" . . .>
    <forward name=".menuRoleManagement:select"
             path="/getRoles.do" redirect="true"/>
    <forward name="default"
             path="/logout.do" redirect="true"/>
</action>
```

If you want to clean the Struts configuration, remove all the action and form bean definitions you do not need. To keep the form bean definitions consistent, remove them in **struts-config.xml** and also remove the details in **forms-config.xml**.

3.4.3. Tiles and Menu Definitions

In addition to changing the form bean definitions, you also need to change the tiles and menu definitions. The most important task is to change the menus. Especially reduce the menu of administration operations displayed with nearly every page.

All the menus are defined in the central file **identity/menu-defs.xml**. The parts required for role management are ".menuRoleManagement", ".contextMenuRoles", ".toolbarRole" and ".toolbarRoles". Keep also some basic definitions: ".menuSelectors", ".menu", ".menuLogout", ".defaultMenubar" and ".startActions". Remove all other definitons.

Then change the default menubar to

```
<definition name=".defaultMenubar">
    <put name="items"
        value=".menuRoleManagement;sep;menuLogout"/>
</definition>
```

3.5. Integrating Management of a New Object Type

This section shows you how to extend the default Web Center application with the management of a new object type – let's say contracts. Contract management comprises:

- Searching contracts.
- Displaying data of a selected contract.
- Modifying contract data.
- Creating contracts.
- Creating menus for contract management.

Since no special widgets are requested and all required operations are present in Web Center, the task only requires editing configuration files, namely **struts-config.xml**, **tiles-defs.xml**, **forms-config.xml, and menu-defs.xml**.

3.5.1. Creating Separate Configuration Files

Web Center offers the possibility of using multiple sets of configuration files **struts-config.xml**, **tiles-defs.xml**, **forms-config.xml, and menu-defs.xml**. In order to separate the customization from the original files as much as possible, we propose to create separate files for this customization task and store them in the directory **WEB-INF/custom/config**. This approach makes it easier to handle version upgrades.

We need to add a reference to the new configuration folder to the context configuration file **webCenter-*domain*.xml**:

```
 <Context docBase="@doc_path@" …
```

```
    <Parameter override="false" name="Struts.DefaultFiles"
        value="struts-config.xml:forms-config.xml:tiles-defs.xml
menu-defs.xml"/>

    <Parameter override="false" name="Struts.Path"
        value="/WEB-INF/config/imports,
               /WEB-INF/config/workflows,
…
               /WEB-INF/config/bo/projects,
               /WEB-INF/custom/config/contracts"/>

    <Parameter override="false" name="ObjectConfig.Path"
        value="/WEB-INF/config/objects-config.xml,
               /WEB-INF/custom/config/objects-config.xml"/>
    <Parameter override="false" name="RendererConfig.Path"
value="/WEB-INF/config/renderers-config.xml"/>

    <!-- Disable session persistence across application reloads and
Tomcat restarts -->
    <Manager pathname=""/>
</Context>
```

In addition, we have to add our custom objects-config.xml file to the configuration (see above).

3.5.2. Searching Contracts

To create a new dialog for searching contracts, the following configuration entries are required:

- The declaration of an action form (in our case, called “contractListForm”) in **struts-config.xml**.
- The definition of an action “/getContracts” in **struts-config.xml**.
- The definition of a page (in our case, “contracts”) containing header, footer, and the “contractsForm” (containing the contract list) in **tiles-defs.xml**.
- The configuration of the contractListForm in **forms-config.xml**.

3.5.2.1. Struts-Config.xml

You must configure the action form and the necessary action mappings.

For the declaration of the action form you must add a form bean with a form name and the type “com.siemens.webMgr.model.DynaLocaleForm” to the form beans section:

```
<form-beans>
    <!-- ActionForm to handle a list contracts request -->
    <form-bean name="contractListForm"
          type="com.siemens.webMgr.model.DynaLocaleForm"/>
</form-beans>
```

Definition of the getContracts-Action:

```
<action-mappings>
<!-- Action to list contracts -->
<action path="/getContracts"
    type="com.siemens.webMgr.controller.action.JSPAction"
    name="contractListForm"
    parameter="jspPage:/WEB-INF/jsp/controller/core/listObjects.jsp;
    elements:contracts;
    selectSingleItem:true;
    excludeTopNodes:2;
    resetPoint:true;
defaultSearchBase:cn=Contracts,cn=Custom,cn=BusinessObjects,${initPar
am['com.siemens.webMgr.ldap.baseDN']};
    defaultFilter:(objectClass=dxrContext);
defaultNameRoot:cn=Contracts,cn=Custom,cn=BusinessObjects,${initParam
['com.siemens.webMgr.ldap.baseDN']};
treeRootNode:cn=Contracts,cn=Custom,cn=BusinessObjects,${initParam['c
om.siemens.webMgr.ldap.baseDN']}">
    <forward name="cancel" path="/logout.do"/>
    <forward name="search" path="/getContracts.do"/>
    <forward name="selectContract"
path="/showContractData.do?ignoreSubmit=true"/>
    <forward name="tile" path=".contracts"/>
    <forward name="updateList" path=".updateContracts"/>
</action>
</action-mappings>
```

The action tag is configured by the attributes **path**, **type**, **name** and **parameter**:

- **Path** - The action path (last part of the URL). The action is accessed by the URL http://**host**:8080/webCenter-*domain*/getContracts.do*.
- **Name** - The name of the form to be used.
- **Type** - The Java action class com.siemens.webMgr.controller.action.JSPAction

- **Parameter** - The Parameter list. JspPage references the JSP to be used to list the objects. The rest of the parameters are passed to the JSP as request parameters.

Five action forwards are defined for the action:

- "**search**" is called if the search button is clicked on the search panel.
- "**cancel**" is called if the request is cancelled.
- **selectContract**" is called if a table entry is selected. The name is derived from the "elements" parameter entry: "select" + "Contract".
- "**tile**" references the whole page named ".contracts" in the **tiles-defs.xml** configuration file.
- "**updateList**" is called in case only the result table of the search is refreshed.

Another Struts action is used to go back to the last search result. The action can be invoked from the main menu or from the toolbar on a contract's overview page.

```
<action path="/lastSelectionContracts"
    type="com.siemens.webMgr.controller.action.JSPAction"
    parameter="jspPage:
        /WEB-INF/jsp/controller/tasks/lastSelection.jsp;
        elements:contracts">
    <forward name="tile" path="/getContracts.do"/>
</action>
```

3.5.2.2. Tiles-defs.xml

Three entries are required in **tiles-defs.xml**: one entry describing the page as a whole, a second entry describing the form dialog, and last not least an entry describing the list form.

The page as a whole is described by the entry named ".contracts":

```
<definition name=".contracts" extends=".base">
<put name="formContent" value=".contractsForm"/>
<put name="textFile" value="custom/contracts/list.html"/>
</definition>
```

The definition extends a common ".base" definition, which is normally used for all pages with standard page layout. The <put> definitions override or set properties in the base definition:

- **formContent** - references the Tiles definition of the content.
- **textFile** - references a static HTML file that contains a comment for the user on how to use the form.

The form content ".contractsForm" is described by the following definition:

```
<definition name=".contractsForm" path="/WEB-
INF/jsp/view/tiles/objectList.jsp">
<put name="action" value="/getContracts.do"/>
<put name="attributes" value="cn;dxrType::contractType;description"/>
<put name="form" value="contractListForm"/>
<put name="listPageSizeItems" value="default"/>
<put name="multiSelectionAllowed" value="true"/>
<put name="objectClasses" value="dxrContext"/>
<put name="objects" value="contracts"/>
<put name="selectedItemsAttribute" value="selectedContract"/>
<put name="size" value="15"/>
<put name="styleClass" value="fullWidthList"/>
<put name="targetId" value="contracts_table"/>
</definition>
```

This contracts form extends the "view JSP" **objectLists.jsp**, which is used for creating and managing the search panel. The specific parameters to be set are:

- action - The default action.
- attributes - The list of LDAP attribute names and labels to be displayed as search attributes.
- form – A reference to the form bean definition in Struts configuration.
- listPageSizeItems – The identifier for the value list of the list's page size selector. The value must match a key listPageSizeItems.<value> in file webCenter.properties which defines the actual value list.
- multiSelectionAllowed – A Boolean to turn on multi-selection in the table.
- objectClasses – The object classes to be searched.
- objects – the element name from object-config.xml extended by plural s
- selectedItemsAttribute – The attribute of the HTTP session where the selected item is stored. It is used to pass the selection to further actions (such as edit or display).
- Size – the number of rows to be displayed.
- styleClass – a reference to a styles.css style.
- targetId – A unique identifier for the result table, usually <objects>_table

The definition for the dynamic list update is described as follows:

```
<definition name=".updateContracts" path="/WEB-
INF/jsp/view/tiles/updateList.jsp">
```

```
<put name="form" value="contractListForm"/>
<put name="listPageSizeItems" value="default"/>
<put name="multiSelectionAllowed" value="true"/>
<put name="objects" value="contracts"/>
<put name="selectedItemsAttribute" value="selectedContract"/>
<put name="size"  value="15"/>
<put name="tableRenderer" value="tableUpdate"/>
<put name="updateMenu" value="true"/>
</definition>
```

The parameters are widely the same as in the .contractsForm definition. In addition, the following parameters exist:

- tableRenderer – The table renderer to be used for update: “tableUpdate”
- updateMenu – a flag indicating the context menu has to be updated when the result table is updated.

3.5.2.3. Forms-config.xml

In the forms configuration file **forms-config.xml** we need to describe the form layout to display the search result in a `<form-bean>` element:

```
<!-- List contracts -->
<form-bean name="contractListForm">
<form-property name="contracts" height="239" width="510"
type="com.siemens.webMgr.model.DirectoryEntryBean[]" readonly="true"
openAction="/openContract.do:ContractMgt.summary"
contextMenu=".contextMenuContracts:ms,list"
secondarySortColumns="$displayName,$parentFolder">
<data-property name="$displayName" label="contract.id"
type="java.lang.String" cellRenderer="link"
width="20%"/>
<data-property name="$parentFolder" type="java.lang.String"
width="20%" cellRenderer="roDN" rendererProperties="includeNodes:[0,-
1];link:true"/>
<data-property name="dxrVersion" label="ldap.attribute.dxrStartDate"
type="java.util.Date" cellRenderer="time"
width="20%" />
<data-property name="description" type="java.lang.String"
cellRenderer="link" width="40%"/>
</form-property>
<form-property name="selectedContract" type="java.lang.String"
```

```
value="-1" transient="true" visible="false"/>
</form-bean>
```

The form property "*contracts*" is a pseudo-property that contains an array of DirectoryEntryBean objects. This is a special class to handle table rows. Its name is build from the object name in the objects-config.xml file, extended by a plural 's'. Height and width define the table size in pixels.

Each DirectoryEntryBean contains the information about one object that is returned by the search (in our case, contracts). The table columns are defined by using the data properties. For each data property, you define the following XML attributes:

- name - the LDAP name of the contract's attribute or a pseudo-attribute (starting with a '$' character).
- type - a Java class that represents the value. In this case, all values are single strings, so the type is "java.lang.String".
- label - the column label definition. It contains a key that references an entry in the language-dependent text properties files. If it is missing, the default key "ldap.attribute.<name>" applies.
- width - the relative column width.
- readonly - if true, the field does not accept any input.
- cellRenderer - as a default, the text field renderer is used for the String type. The type "link" here does not permit editing the text but returns the index of the row when clicked. In our sample, the ldap attribute dxrVersion is used to hold the contract's start date. Thus, the field renderer "time" has to be used.

The form property itself supports some additional attributes that apply to the entire table:

- openAction – defines the action to be called when a table row is selected.
  In our sample, the open action is defined as follows in the struts-config.xml file:

  ```
  <action
  path="/openContract"type="com.siemens.webMgr.controller.action.JSP
  Action"
  parameter="jspPage:/WEB-
  INF/jsp/controller/core/openEntry.jsp;entryType:contract">
  <forward name="tile" path="/showContractData.do"/>
  </action>
  ```

  This action sets the session variable holding the selection and then calls the action to display the contract using the "tile" forward.

- contextMenu – references a definition of a context menu being displayed on the results table (see context menu definition below).

- secondarySortColumns – specifies the secondary sort columns for each column; the primary sort criterion is the column itself.

3.5.3. Creating Menus and Static HTML Pages

At this point we still have some problems with our solution:

- We do not want to enter the requested URL manually, but select the action from a menu instead. Therefore, we need to add a menu entry.
- We have to add a static html file with information what to do on the page. If the file is missing, we will get an error message:

```
“The requested resource ((/WebCenter/WEB-
INF/classes/resources/languages/de/custom/contracts/list.html) is
not available.” Reason: the static HTML page
custom/contracts/list.html is missing.
```

3.5.3.1. Creating a Menu for Contract Processing

The menu is described in a file menu-defs.xml. We put the file into the same folder as the forms-config.xml, struts-config.xml, and tiles-defs.xml files.

The menu definition for the pull-down menu (from the menu bar) is

```
<definition name=".menuContractManagement" extends=".menu">
<put name="accessPolicyKey" value="ContractMgt"/>
<put name="msgPrefix"       value="contracts"/>
<put name="items" value="select:/getContracts.do;
lastSelection:/lastSelectionContracts.do:contractsFilter;
create:/createContract.do;
sep;
summary:/showContractData.do:selectedContract;
modify:/editContractData.do:selectedContract:mod"/>
</definition>
```



The menu .menuContractManagement has to be referenced from the .defaultMenubar definition being located in the /WEB-INF/config/identity/menu-defs.xml file

3.5.3.2. Creating Contract Context Menus

To create a context menu for the result list of contract search, the following menu definition is added to our menu-defs.xml file:

```
<definition name=".contextMenuContracts">
<put name="accessPolicyKey" value="ContractMgt"/>
<put name="msgPrefix" value="ctx.objects"/>
<put name="items"     value="summary;modify;
list;export"/>
</definition>
```

To get a context menu working, it has to be referenced from the form definition "*contractListForm*" in its contextMenu attribute.

In addition, an associated action "/contextMenuContracts" has to be defined in the struts-config.xml file, to connect the items with the actions:

```
<action path="/contextMenuContracts"
    type="com.siemens.webMgr.controller.action.JSPAction"
    parameter="jspPage:/WEB-INF/jsp/controller/tasks/contextMenu.jsp;
    selectedObject:com.siemens.webMgr.selectedContract">
<forward name="export"
path="/exportContractList.do"/>
<forward name="modify"                path="/editContractData.do"/>
<forward name="summary"               path="/showContractData.do"/>
</action>
```

This action has to use the JSP contextMenu.jsp.

3.5.3.3. Extending the Message Files

Do not forget to add the appropriate label keys and message texts into the message properties files. By default, these are:

- "WEB-INF/classes/resources/languages/en/text.properties" and
- "WEB-INF/classes/resources/languages/de/text.properties".

It's a good idea to put custom messages into separate files named

- "WEB-INF/classes/resources/languages/en/text_en.properties" and
- "WEB-INF/classes/resources/languages/de/text_de.properties".

The following sample shows English message texts:

```
# The custom definitions start here

contract.id            = Contract
```

```
contracts.title         = Contracts
contracts.select        = Select contract
contracts.create        = Create contract
contracts.summary       = Show contract
contracts.modify        = Modify contract
contracts.lastSelection = &Last selection list: {30s}

searchPanel.title.contracts = Search contracts
form.title.data.contract    = Contract data
```



The message id contract.id is defined as a label in the forms-config.xml file. The main menu definitions above define message key prefix “contracts”. The message key for a menu item label is then comprised of prefix and item name, separated by a dot:

Label=<prefix>.<item>, for example “`contracts.select`“.

Note also that we defined additional messages for the title of the search panel for the contracts list, the overview form title and others which we will use later on.

3.5.3.4. Creating Static HTML Texts

The static HTML pages normally contain textual comment shown near the top of the page content beneath the header. These pages must be created for all supported languages; by default, English and German. We place them in the folders “WEB-INF/classes/resources/languages/en/custom/contracts” and “WEB-INF/classes/resources/languages/de/custom/contracts”.

As templates to copy, you can use the corresponding context text files, for example, “bo/contexts/list.html”.

3.5.4. Extending the Navigation History Control

We announce our new object type to the navigation history control in order to automatically get visited contract pages added to the navigation history. In file **WEB-INF/config/webCenterCustom.properties**, append the object type to the property with name “navigationHistory.conf.objects”:

```
navigationHistory.conf.objects = Contract
```

3.5.5. Displaying Contract Data

For this sample, we use a simplified variant of the existing form bean definition for the overviewContextForm.

3.5.5.1. Definitions in forms-config.xml

The dialog to display contract data is defined as follows:

```
<form-bean name="overviewContractForm">
<form-property name="$parentFolder" type="java.lang.String"
transient="true"
       fieldRenderer="roDN" rendererProperties="includeNodes:[0,-1]"
       width="600" spanX="4" readonly="true" y="0"/>
<form-property name="$displayName" label="contract.id"
type="java.lang.String" fieldRenderer="boldText" readonly="true"
       width="100%" y="+1"/>
<form-property name="description" type="java.lang.String"
readonly="true"
       width="600" spanX="4" y="+1"/>
<form-import name="subjectModifyOrders"/>
<form-property-group name="operationalAtts" y="+1" spanX="4"
groupRenderer="groupWithLabel"/>
<form-property name="owner" type="java.lang.String[]"
fieldRenderer="userLinks" readonly="true"
       width="100%" y="+1"/>
<form-property name="dxrVersion" type="java.util.Date"
label="ldap.attribute.dxrStartDate" readonly="true"
   width="180" y="+1"/>
<form-property name="dxrState" type="java.lang.String"
readonly="true"
       width="180" y="+1"/>
</form-bean>
```

The $parentFolder pseudo-property contains the parent folder's DN. It is rendered by the special fieldRenderer roDN. includeNodes defines which nodes of the DN are displayed. The first number gives the position of the first RDN, in this case 0 for the first RDN. The second number defines the position of the last RDN to display; negative numbers count from the end, so -1 is the second but last RDN. Note that the topmost node, the domain, is already stripped off by default due to setting dn.excludeTopNode to 1 in webCenter.properties and config.js.

The $displayName pseudo-property contains the value of the naming attribute, that is defined in the object description for contracts. In our case it is the contract's cn. The cn's default-label ldap.attribute.cn, evaluating to 'name', is overwritten with the label definition "contract.id", evaluating to "contract" or "Vertrag".

The description attribute is a standard X500 attribute, holding the contract's description.

The property "subjectModifyOrders" is a pseudo-property. It is used to display active orders for subject modifications.

The property “owner” holds the contract owners.

The property “dxrVersion” is used in this sample to display the contract’s start date.

Last-not-least the property dxrState holds the object’s state.

3.5.5.2. Definitions in struts-config.xml

We define a new action showContractData the following way:

```
<action path="/showContractData"
type="com.siemens.webMgr.controller.action.JSPAction"
name="overviewContractForm"
parameter="jspPage:/WEB-INF/jsp/controller/ core/showData.jsp;
objectVar:selectedContract;
object:${sessionScope['com.siemens.webMgr.selectedContract']}">
<forward name="tile" path=".overviewContract"/>
</action>
```

We can use the existing “showData.jsp” to display the contract data. The DN of the selected contract is read from the session attribute ‘com.siemens.webMgr.selectedContract”. This corresponds to the element definition ‘contracts’ in the Struts configuration of the action “getContracts”.

Now we have to add the used form bean “overviewContractForm” to the form beans list at the top of the file:

```
<form-beans>
    …
    <form-bean name=" overviewContractForm"
          type="com.siemens.webMgr.model.DynaLocaleForm"/>
</form-beans>
```

3.5.5.3. Definitions in tiles-defs.xml

We need two entries in **tiles-defs.xml**: one that describes the page as a whole and one that describes the form dialog.

The page as a whole is described in the “.overviewContract” entry:

```
<definition name=".overviewContract" extends=".base">
```

```
<put name="formContent" value=".overviewContractForm"/>
<put name="textFile" value="custom/contracts/overview.html"/>
<put name="readonly" value="true"/>
</definition>
```

The following "put" parameters are to be set:

- formContent – a reference to the Tiles definition form for the content.
- textFile – a reference to a static HTML file that contains the comment for the user on how to use this form.
- readonly - if true, the form does not accept any input.

The referenced form content is defined in the following “.overviewContractForm” entry:

```
<definition name=".overviewContractForm" path="/WEB-
INF/jsp/view/tiles/form.jsp">
<put name="action"      value="/showContractData.do"/>
<put name="formTitles"  value="data.contract"/>
<put name="formToolbar" value=".toolbarContract"/>
<put name="styleClass"  value="overviewForm"/>
<put name="readonly"    value="true"/>
</definition>
```

and needs these "put" parameters:

- action – a reference to the Struts action defined above.
- formTitles – the message key for the form title.
- formToolbar – the toolbar definition name from menu-defs.xml.
- styleClass – a reference to a styles.css style.
- readonly - the value true sets the read-only mode.

3.5.6. Modify a Contract

For contract modification, we need the appropriate form beans and actions for edit and save.

3.5.6.1. Definitions in struts-config.xml

We need to:

- Define a form bean for the modification form (modifyContractForm)
- Create an action to edit a contract: “editContractData”. We use “showData.jsp” to fill the form with the existing attributes.

- Create a second action to edit a contract: “editContractData2”. This action is required when the owner is edited. In this case, another dialog to select the owner is displayed. After selecting the owner, process flow returns to “editContractData2”, being configured with clearForm:false. Thus, edits added to the form prior to editing the owner are not cleared when the modifyContractForm is displayed again. We use “showData.jsp” to fill the form with the existing attributes.
- Create an action to store the contract in the Identity domain: “storeContractData”. We use “storeData.jsp” to save the modifications.

Here are the corresponding definitions:

```
<action path="/editContractData"
type="com.siemens.webMgr.controller.action.JSPAction"
name="modifyContractForm"
parameter="jspPage:/WEB-INF/jsp/controller/ core/showData.jsp;
operation:modify;
object:${sessionScope['com.siemens.webMgr.selectedContract']}">
<forward name="disallowed" path="/showContractData.do"
redirect="true"/>
<forward name="tile" path=".modifyContract"/>
</action>

<action path="/editContractData2"
type="com.siemens.webMgr.controller.action.JSPAction"
name="modifyContractForm"
parameter="jspPage:/WEB-INF/jsp/controller/ core/showData.jsp;
operation:modify;
object:${sessionScope['com.siemens.webMgr.selectedContract']};
clearForm:false;attributes:!owner">
<forward name="disallowed" path="/showContractData.do"
redirect="true"/>
<forward name="tile" path=".modifyContract"/>
</action>

<action path="/storeContractData"
type="com.siemens.webMgr.controller.action.JSPAction"
name="modifyContractForm"
parameter="jspPage:/WEB-INF/jsp/controller/ core/storeData.jsp;
objectVar:selectedContract;
object:${sessionScope['com.siemens.webMgr.selectedContract']}">
<forward name="owner"
path="/getUsersForObject.do?mainAction=/editContractData2.do&amp;forw
```

```
ard=owner&amp;attributes=dn.owner[]"/>
<forward name="next" path="/showContractData.do" redirect="true"/>
<forward name="tile" path=".modifyContract" />
</action>
```

The contract object is again identified by the session attribute “com.siemens.webMgr.selectedContract”.



The action forward ‘owner’ at the ‘storeContractData’ action is called when the button to select a new owner is pressed. This is a property of the field renderer ‘userLinks’ being used to render the owner. The path attribute contains the action ‘getUsersForObject’, being used to search and select the owner. After this, process flow returns to the mainAction ‘editContractData2’, as already mentioned. The attributes-property defines that the selected user’s dn is stored in the contract’s owner attribute. The array brackets are used since here owner is multi-valued, i.e. the new or modified owner is added to the array of owners.

Finally, we have to add the used form bean “modifyContractForm” to the form beans list at the top of the file:

```
<form-beans>
    …
    <form-bean name="modifyContractForm"
          type="com.siemens.webMgr.model.DynaLocaleForm"/>
</form-beans>
```

3.5.6.2. Definitions in tiles-defs.xml

We need the two entries in **tiles-defs.xml** for the page as a whole and for the form dialog:

```
<definition name=".modifyContract" extends=".base">
<put name="formContent" value=".modifyContractForm"/>
<put name="textFile" value=" custom/contracts/modify.html"/>
</definition>

<definition name=".modifyContractForm" path="/WEB-
INF/jsp/view/tiles/form.jsp">
<put name="action" value="/storeContractData.do"/>
</definition>
```



We use the action “storeContractData” to save the modifications. This

action is called after the form is submitted by clicking Save or Cancel.

3.5.6.3. Definitions in forms-config.xml

For the modification form layout, we take the appropriate context modification form “modifyContextForm” as a template and change it:

```
<form-bean name="modifyContractForm"
buttons="dueDate.modify;enterConfirm:application.submit;resetConfirm:
application.cancel">
<form-property name="$parentFolder" type="java.lang.String"
transient="true"
fieldRenderer="roDN" rendererProperties="includeNodes:[0,-1]"
width="600" spanX="4" readonly="true" y="0"/>
<form-property name="cn" type="java.lang.String" mandatory="true"
label="contract.id"
   width="100%" y="+1"/>
<form-property name="description" type="java.lang.String"
   width="600" spanX="4" y="+1"/>
<form-property-group name="operationalAtts" y="+1" spanX="4"
groupRenderer="groupWithLabel"/>
<form-property name="owner" type="java.lang.String[]"
fieldRenderer="userLinks"
   width="100%" y="+1"/>
<form-property name="dxrVersion" type="java.util.Date"
label="ldap.attribute.dxrStartDate"
   width="180" y="+1"/>
<form-property name="dxrState" type="java.lang.String"
readonly="true"
   width="180" y="+1"/>
</form-bean>
```



The form needs a Save and a Reset button. The dueDate.modify button is a button to enter a due date, to create a ticket for the modification. Since forms for simply viewing and for editing are defined separately, they may show a different layout with different attributes.

3.5.7. Creating a Contract

For creating a contract, we need to define the form beans for the creation form and actions to fill the form with default data and to create the contract object.

3.5.7.1. Definitions in struts-config.xml

We need to:

- Define a form bean for the creation form (createContractForm)
- Create an action “createContract” that uses “createObject.jsp” to fill the form with default data.
- Create an action “createContract2” that uses “createObject.jsp” to create the account object.

```
<action path="/createContract"
    type="com.siemens.webMgr.controller.action.JSPAction"
    name="createContractForm"
    parameter="jspPage:
        /WEB-INF/jsp/controller/core/createObject.jsp;
        elements:contracts">
    <forward name="tile" path=".createContract"/>
</action>

<action path="/createContract2"
    type="com.siemens.webMgr.controller.action.JSPAction"
    name="createContractForm"
    parameter="jspPage:
        /WEB-INF/jsp/controller/core/createObject.jsp;
        clearForm:false;
        elements:contracts;
        selectedItemsAttr:selectedContract">
    <forward name="cancel" path="/getContracts.do"
        redirect="true"/>
    <forward name="owner"
        path="/getUsersForObject.do?
              mainAction=/createContract2.do&amp;
              forward=owner&amp;attributes=dn.owner[]"/>
    <forward name="show" path="/showContractData.do"
        redirect="true"/>
    <forward name="tile" path=".createContract"/>
</action>
```

The “createObjects.jsp” expects the following parameters:

- elements – a reference to an entry in objects-config.xml. The object id is generated by removing the last character ("s" here). Thus, we reference a new entry ‘contract’ in the **object-configs-xml** file. It has the following definition:

```
<object id="contract"
   nodeClass= "siemens.dxr.organization.nodes.SvcContext"
   set="Contexts"
   dirClass="Contract"
   def="false"
   policyType="Contract" />
```



The name of the contract's object description (in our sample 'Contract') has to be used for dirClass and policyType. />

- clearForm - if set to false, the existing form contents is not cleared.
- selectedItemsAttr - In this session attribute, the DN of the created object is stored for processing in the next actions.

Once again, add the form bean to the list of form beans at the top of the file

```
<form-beans>
    …
    <form-bean name="createContractForm"
          type="com.siemens.webMgr.model.DynaLocaleForm"/>
</form-beans>
```

3.5.7.2. Definitions in tiles-defs.xml

We need the two entries in **tiles-defs.xml** for the page as a whole and for the form dialog:

```
<definition name=".createContract" extends=".base">
    <put name="formContent" value=".createContractForm"/>
    <put name="textFile"
         value="custom/contracts/create.html"/>
</definition>
<definition name=".createContractForm"
            path="/WEB-INF/jsp/view/tiles/form.jsp">
    <put name="action" value="/createContract2.do"/>
</definition>
```

3.5.7.3. Definitions in forms-config.xml

The form to create contracts can be derived from the modification form. The parent folder is made editable and assigned the field renderer "dn". The due date button label is changed from dueDate.modify to dueDate.create. The dxrType attribute is omitted since it is read-

only.

```
<form-bean name="createContractForm"
           buttons="dueDate.create;
                    enterConfirm:application.submit;
                    resetConfirm:application.cancel">
    <form-property name="$parentFolder"
        type="java.lang.String" transient="true"
        fieldRenderer="dn"
        rendererProperties="includeNodes:[0,-1]"
        width="600" spanX="4" y="0"/>
    <form-property name="cn"
        type="java.lang.String" mandatory="true"
        label="contract.id"
        width="100%" y="+1"/>
    <form-property name="description"
        type="java.lang.String"
        width="600" spanX="4" y="+1"/>
    <form-property-group name="operationalAtts"
        y="+1" spanX="4"
        groupRenderer="groupWithLabel"/>
    <form-property name="owner"
        type="java.lang.String[]"
        fieldRenderer="userLinks"
        width="100%" y="+1"/>
    <form-property name="dxrVersion"
        type="java.util.Date"
        label="ldap.attribute.dxrStartDate"
        width="180" y="+1"/>
</form-bean>
```

3.5.8. Configuring Access Policies

Creating contracts is subject to access control. As a result, you need to configure access policies that allows the appropriate users (for example, members of group “ContractAdmins”) to create, read and modify contracts.

The name of the object description of contracts (in our sample ‘Contract’) has to be used as object type in the access policies.

4. Renderers

4.1. Overview

Any user interface control in any form appearing in a Web Center JSP page must be converted to a standard HTML tag before it can be inserted into the resulting HTML page which is then sent to the client browser. These converters are called renderers.

The standard renderers are defined in file **WEB-INF/config/renderers-config.xml**. Each renderer has the following attributes:

- **id** – The renderers unique id; required.
- **type** – The Java class of the properties the renderer is used to display or edit, like “java.lang.String”, “java.lang.String[]” or “java.util.Map”.Default: “java.lang.String”.
- **className** – The renderer class. For each supported type, Web Center provides a special renderer class which handles the conversion of properties of that type to HTML and Javascript.Default: “com.siemens.webMgr.taglib.view.renderers.BasicRenderer”.

The conversion is based on HTML and Javascript code snippets which are assigned as attributes to each renderer. There are four different types of snippets:

- **defURL –** The default HTML code snippet for displaying the property in an HTML form. This attribute is required.
- **altURL –** The HTML code snippet for displaying the property in edit mode in an HTML form. Default: **defURL**.
- **jsURL –** The default Javascript code snippet for displaying the property in an HTML table cell.
- **roJsURL –** The Javascript code snippet for displaying the property in read-only mode in an HTML table cell. Default: **jsURL**.

Renderer properties can be used to pass variable values to a renderer class and renderer snippets. While each renderer class has a predefined set of supported properties, renderer snippets may use additional properties which are referenced in the snippets by their name. If a renderer property value is a message key, it is replaced by the actual localized message text at runtime.

A renderer can also inherit code snippets and properties from another renderer, and reassign just the ones that are different:

- **extends** – The id of the parent renderer. The renderer inherits type, class, snippets and properties.

4.2. Custom File Overview

In this section, we assume that custom files are located below the application’s root folder in the following way:

```
resources
    custom
        messages
            de
                messages.js
            en
                messages.js
        scripts
            custom.js
        styles
            small
                styles.css
            medium
                styles.css
            large
                styles.css
WEB-INF
    classes
        resources
            languages
                de
                    text_de.properties
                en
                    text_en.properties
    config
        webCenterCustom.properties
    custom
        config
            converters.properties
            defaultRenderers.properties
            messages.properties
            objects-config.xml
            renderers-config.xml
        jsp
            scriptsAndStyles.jsp
        snippets
            ...
```

You may choose different folders and names for your custom files. Files with fixed names and locations are printed in bold font. See the “Customization Files” for details.

4.3. Customizing Renderers

You can write a new renderer from scratch, providing your own code snippets and all. On the other hand, you can derive a new renderer from another one by assigning different values to its renderer properties, or by reassigning one or more of its code snippets.

Samples for each alternative are given below. As always, you should try to get away with the simplest solution, reusing as much as possible.

When adding new components like renderer definitions, code snippets, message texts and stylesheet classes, it's good to make the new components easily distinguishable from the standard components, either by assigning names that start with a specific prefix (like "custom"), by using dedicated naming scopes, or by putting them in separate folders. This makes it easier to migrate the extensions to later product versions.

We also strongly recommend to keep customizations in separate files and to keep changes to the files delivered with Web Center at a minimum.

The next sections describe in detail how to write customized renderers. To get a first feeling of how it works, take a look at the simple samples in the last section.

4.4. Creating a New Renderer

Creating a new renderer from scratch involves the following steps:

- Writing HTML and/or Javascript code snippets.
- Defining the renderer in a custom renderers definition file.

In addition to that, it may further involve

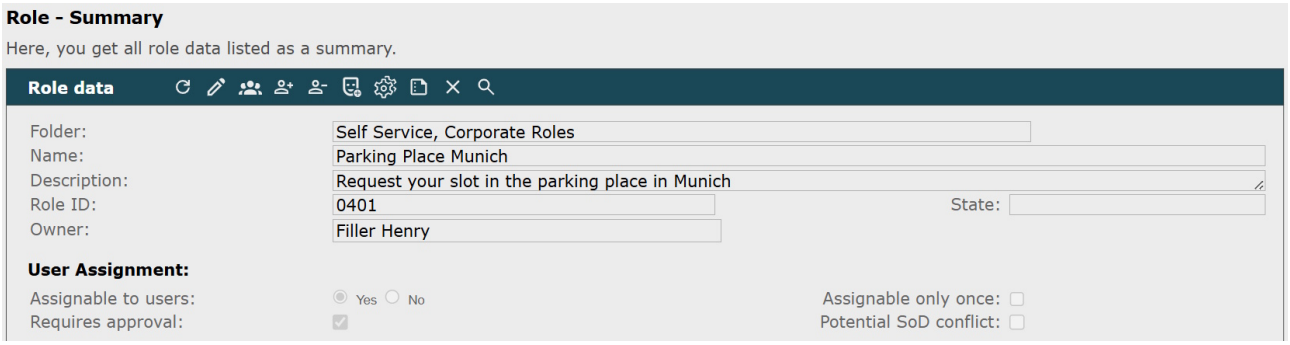
- Adding messages to text_<language>.properties files.
- Adding style definitions to stylesheets.
- Adding images.
- Adding code to Javascript source files.

We will walk through the steps by means of a sample.

4.4.1. Sample Renderer Use Case

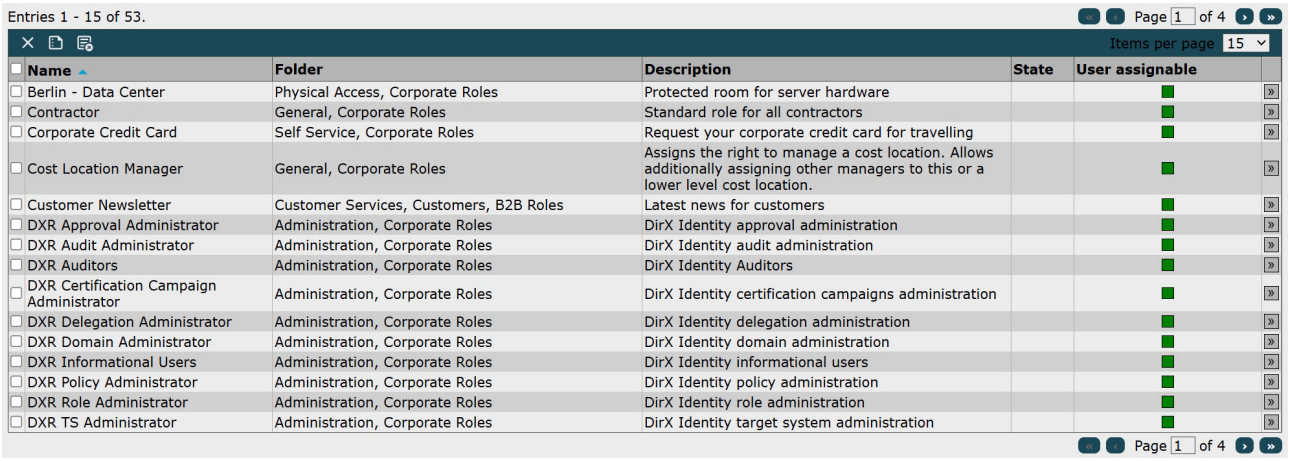
Let's say we want to write a new renderer for boolean properties that displays two radio buttons for the property value, one for "true" and one for "false". The radio button for "true" should be checked if the property value is "true", while the radio button for "false" should be checked otherwise. The button labels, i.e. the text to be displayed along with each button, should be localizable and have the same color and font as other labels. The renderer is to be used in HTML forms in read-only as well as in editable mode.

The following figure shows how this renderer displays the user assignable flag of a role:



In table columns for boolean properties, green squares should indicate “true”, while red squares indicate “false”. One can change color and value by clicking on a square.

The following figure shows how this renderer displays the user assignable flag of roles:



4.4.2. Renderer Definition

We define the new renderer in file **WEB-INF/custom/config/renderers-config.xml**:

```
<renderer id="customBoolean" type="java.lang.Boolean"
       defURL="/WEB-INF/custom/snippets/boolean/roBoolean.htm"
       altURL="/WEB-INF/custom/snippets/boolean/rwBoolean.htm"
       jsURL="/WEB-INF/custom/snippets/boolean/boolean.js">
    <renderer-property name="labelTrue"  value="custom.boolean.yes"/>
    <renderer-property name="labelFalse" value="custom.boolean.no"/>
</renderer>
```

The renderer id starts with “custom” in order to make it clear that it is a customized renderer.

The type “java.lang.Boolean” is determined by the types of the properties the renderer is used to display. We don’t have to assign a renderer class name since the default class will do.

The code snippet assigned to “defURL” will be used to display boolean properties in read-only mode in HTML forms, while the snippet assigned to “altURL” is for displaying the properties in editable mode.

The code snippet assigned to “jsURL” will be used to display boolean properties in table cells. The renderer applies to read-only and read-write properties.

The custom renderer properties “labelTrue” and “labelFalse” accept the message keys for the button labels. The properties will be used in the code snippets.

4.4.3. Code Snippets

4.4.3.1. HTML Form Field - Read-Only Mode

The HTML code snippet for read-only mode is defined in file **roBoolean.htm**:

```
<input type="radio" name="${name}" id="${id}_true" value="true"
       disabled="disabled"/>
<span class="customBooleanLabel">${labelTrue}</span>
<input type="radio" name="${name}" id="${id}_false" value="false"
       disabled="disabled"/>
<span class="customBooleanLabel">${labelFalse}</span>
<script>custom.initRadioButton("${id}","${value.jsd}");</script>
```

The snippet adds two radio buttons with the same name, the property name. The first button is used for property value “true”, the second one for “false”. The buttons are disabled since the snippet is for read-only pages only.

Each radio button is followed by its label. The labels reference the custom renderer properties “labelTrue” and “labelFalse”, respectively. The labels are enclosed in “span” elements that define their CSS class name.

The Javascript scriptlet at the bottom of the snippet checks if the actual property value is “true” or “false”, and activates the corresponding radio button.

The snippet references (via ${...} expressions) some renderer properties that are replaced at runtime by their actual values:

- **name** – The property name.
- **id** – A unique HTML element identifier which is essentially the concatenation of the form name and the property name, separated by an underscore. The id can be used in Javascript code to find HTML elements defined in the code snippet.
- **value** – The property value. The extension “.jsd” indicates to Web Center that the value is used within Javascript code and is enclosed in double quotes. Web Center uses this information to properly escape the value.
- **labelTrue** and **labelFalse** – The custom renderer properties for the label message keys. At runtime, the variables are replaced by the corresponding localized message texts.

The HTML code generated at runtime may for example look like

```
<input type="radio" name="dxrUserAssignmentPossible"
    id="overviewRoleForm_dxrUserAssignmentPossible_true"
    value="true" disabled="disabled"/>
<span class="customBooleanLabel">Yes</span>
<input type="radio" name="dxrUserAssignmentPossible"
    id="overviewRoleForm_dxrUserAssignmentPossible_false"
    value="false" disabled="disabled"/>
<span class="customBooleanLabel">No</span>
<script>custom.initRadioButton("${id}","${value.jsd}");</script>
```

4.4.3.2. HTML Form Field - Editable Mode

The HTML code snippet for read-write mode is defined in file **rwBoolean.htm**:

```
<input type="radio" name="${name}" id="${id}_true" value="true"/>
<span class="customBooleanLabel">${labelTrue}</span>
<input type="radio" name="${name}" id="${id}_false" value="false"/>
<span class="customBooleanLabel">${labelFalse}</span>
<script>custom.initRadioButton("${id}","${value.jsd}");</script>
```

The snippet is almost identical to the read-only snippet. The only difference is that the radio buttons are not disabled in order to allow for changing their values.

4.4.3.3. HTML Table Cell

The Javascript code snippet for displaying boolean properties in table cells is defined in file **boolean.js**:

C_BO("${value}")

“C_BO” is a unique short identifier for the Javascript method that actually renders the values.

4.4.4. Javascript Functions

The above renderers invoke Javascript functions to check a radio button and to render values in table cells.

We choose a separate naming scope for our extensions, namely “custom”, in order to avoid naming conflicts with existing Javascript code.

4.4.4.1. Scriptlet Function

The code to check a radio button is:

```
var custom = {
    initRadioButton: function(id,value) {
        var button = value === "true";
        document.getElementById(id+"_"+button).checked = true;
    },
    ...
};
```

4.4.4.2. Table Cell Renderer Function

The task of a table cell renderer function is to create the DOM representation for the property value. This may be just a single DOM element, or a DOM tree. The renderer does not add the DOM representation to the table cell but just returns the DOM element or the root of the DOM tree to the caller.

Renderer functions are invoked with a couple of predefined arguments:

- **p** – The DOM element representing the table cell.
- **n** – The request parameter name used to send modified property values to the server.
- **m** – Indicates whether to generate cell content for read-only (m === READONLY) or read-write mode (m === READWRITE). READONLY and READWRITE are Javascript constants in global scope.
- **v** – The current property value (after editing it), or null.
- **phValue** – The original property value, as sent from the server.

In addition to that, some complex renderers have additional arguments.

The sample code to render boolean values in read-only mode is

```
var custom = {
    ...
    renderBoolean: function(p,n,m,v,phValue) {
        var isTrue = (v || phValue) === "true";
        var square = document.createElement("div");
        square.className = isTrue ? "customYesCell" : "customNoCell";
        return square;
    }
}
```

The sample renderer creates a new “div” element and assigns it a CSS class depending on the value of the boolean property.

In order to extend the above function to read-write mode, we first have to understand how tables work in Web Center. If a table is newly displayed, Web Center invokes the renderer functions for each table cell in read-only mode with arguments

```
(m,v,phValue) === (READONLY,null,<initial value>).
```

If one clicks on a table cell, Web Center invokes the renderer function of the cell in read-write mode with arguments

```
(m,v,phValue) === (READWRITE,null,<initial value>).s
```

On value changes, the renderer must store the new value in attribute “value” of a DOM element (created by the renderer) with id “<cell id>_edit”. Then, if one clicks somewhere outside the table cell, Web Center persists the possibly modified value in a hidden field and invokes the renderer function again in read-only mode, but this time with arguments

```
(m,v,phValue) === (READONLY,<modified value>,<initial value>).
```

If the cell is clicked again, the renderer function is called in read-write mode:

```
(m,v,phValue) === (READWRITE,<modified value>,<initial value>).
```

Before Web Center invokes a renderer function it deletes the content of the table cell. The renderer, therefore, has to recreate the table content on each invocation.

In our sample, we want let the users change a boolean value by clicking on a square, but not by clicking into the cell somewhere outside the square. This poses a slight difficulty since the “div” elements representing the squares do not propagate clicks to the table cell. Thus, a click on a square will not lead to a call of the cell’s renderer function. On the other hand, a click somewhere outside the square will invoke the renderer function but should have no effect at all.

Therefore, we first assign an onclick handler to the square:

square.onclick = custom.squareClicked;

The onclick handler gets the DOM element representing the table cell, sets its “squareClicked” attribute to “true”, and propagates the click to the table cell by firing a corresponding event, which will lead to a call of the cell’s renderer function in read-write mode:

```
    var p = getSource(e).parentNode;
    p.squareClicked = true;
    fireEvent("click",p);
```



The handler uses the utility functions “getSource” and “fireEvent” defined in one of Web Center’s Javascript files.

Now, whenever our renderer function is called in read-write mode, we have to check the value of the cell’s “squareClicked” attribute. If its value is “true”, the call was caused by clicking on the cell’s square, and we must invert the value of the boolean property and reset the attribute. Otherwise, we don’t change the value:

```
    if (p.squareClicked) {
        isTrue = !isTrue;
        p.squareClicked = false;
    }
```

Finally, we save the current boolean value in the square’s “div” element, and set its id to the cell’s id plus “_edit”:

```
    square.id = p.id + "_edit";
    square.value = isTrue ? "true" : "false";
```

This will let Web Center store the new value in a hidden field when the cell changes back to read-only mode. Note that the lifetime of the “div” element is temporary since the cell content is cleared each time the renderer is invoked.

The complete renderer code is:

```
var custom = {
    ...
    squareClicked: function(e) {
        var p = getSource(e).parentNode;
        p.squareClicked = true;
        fireEvent("click",p);
        return false;
    },

    renderBoolean: function(p,n,m,v,phValue) {
        var isTrue = (v || phValue) === "true";
        var square = document.createElement("div");
```

```
        if (m === READWRITE) {
            if (p.squareClicked) {
                isTrue = !isTrue;
                p.squareClicked = false;
            }
            square.id = p.id + "_edit";
            square.value = isTrue ? "true" : "false";
        }

        square.className = isTrue ? "customYesCell" : "customNoCell";
        square.onclick = custom.squareClicked;
        return square;
    }
}
```

4.4.4.3. Registering Renderers

Web Center provides a Javascript object named **renderers** which is responsible to decide which renderer to take for which table cell. The object keeps a map of renderer ids to renderer functions. Initially, the map is filled with the standard cell renderers provided with Web Center.

Custom cell renderers must be registered with the **renderers** object in order to get added to the map. The object provides a registration method:

- **register(renderer ...)** – Registers one or more cell renderers. The function accepts one or more arguments. Each argument is an array of three items:
  - **id –** The renderer id..
  - **type –** The renderer type, one of **args**, **noArgs** and **noValue**. Use **args** for renderers with additional arguments besides the cell value, **noArgs** for renderers whose only argument is the cell value, and **noValue** for renderers without any argument at all.
  - **f –** The renderer function (function).

The only argument of our sample custom renderer is the cell value. Thus, we register the renderer with:

```
renderers.register(["C_BO","noArgs",custom.renderBoolean]);
```

4.4.4.4. Making the Code Available to the Client

Add an include statement for your custom script file to the custom JSP **WEB-INF/custom/jsp/scriptsAndStyles.jsp**:

```
<script src="resources/custom/scripts/custom.js"></script>
```

4.4.5. Message Texts

The renderer uses two message keys for the button labels. We have to assign message texts to the keys for each supported language, like for example English and German.

4.4.5.1. English

File: **WEB-INF/classes/resources/languages/en/text_en.properties**

Messages:

```
custom.boolean.yes = Yes
custom.boolean.no  = No
```

4.4.5.2. German

File: **WEB-INF/classes/resources/languages/de/text_de.properties**

Messages:

```
custom.boolean.yes = Ja
custom.boolean.no  = Nein
```

4.4.6. Stylesheets

The renderer assigns CSS class names to the button labels and to the div elements for the red and green squares. We have to define the classes for each supported stylesheet variant, like for example medium and large.

4.4.6.1. Medium Font Size

File: **resources/custom/styles/medium/styles.css**

Definitions:

```
.customBooleanLabel, .customBooleanRwLabel {
    font:11px Arial,sans-serif;
    padding-left:2px;
    padding-right:2px;
}
.customBooleanLabel {
```

```
       color:#666666;
}
.customBooleanRwLabel {
    color:#003366;
}

.customYesCell, .customNoCell {
    border:1px solid black;
    width:11px;
    height:11px;
    font-size:1px;
    position:relative;
}
#msie .customYesCell, #msie .customNoCell {
       display:inline
}
.customYesCell {
    background-color:green;
}
.customNoCell {
    background-color:red;
}
```

4.4.6.2. Large Font Size

File: **resources/custom/styles/large/styles.css**

Definitions:

The styles are identical to the medium ones, except for a larger font size, height and width.

```
.customBooleanLabel, .customBooleanRwLabel {
    font:12px Arial,sans-serif;
       ...
}

.customYesCell, .customNoCell {
       ...
    width:12px;
    height:12px;
       ...
```

```
}
```

4.4.6.3. Making the Styles Available to the Client

Import the core tag library and add an include statement for your custom script file to the custom JSP **WEB-INF/custom/jsp/scriptsAndStyles.jsp**:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<link rel="StyleSheet" type="text/css"
      href="resources/custom/styles/
          <c:out
value="${sessionScope['com.siemens.webMgr.style']}"/>
          /styles.css">
```

4.5. Extending Existing Renderers

4.5.1. Reassigning Property Values

The simplest way to create a new renderer is to extend another one, assign a new id and modify some of its property values.

For example, the renderer “objectSearch displays a text field and an action button. It is used to select the DN of an object that should be assigned to a property of another object, e.g. to the approval workflow link of a role. The assigned object’s “cn” is then displayed in the text field. The name of the display property is defined by the value of the renderer property “pattern”:

```
<renderer id="objectSearch”
    className="..."
    defURL="/WEB-INF/snippets/textField/roTextField.htm"
    altURL="/WEB-INF/snippets/textField/roUserSearch.htm"
    jsURL="/WEB-INF/snippets/textField/userSearch.js">
  <renderer-property name="action"
                     value="/selectObject.do"/>
  <renderer-property name="changeTooltip"
                     value="application.tooltip.changeValue"/>
  <renderer-property name="deleteTooltip"
                     value="application.tooltip.deleteValue"/>
  <renderer-property name="pattern"
                     value="${cn}"/>
  <renderer-property name="selectObjectTooltip"
```

```
                    value="application.tooltip.selectObject"/>
</renderer>
```

But it's not always appropriate to display the common name of the assigned object. Organizational units, for example, don't even have a common name. For links to organizational units, therefore, we define a new renderer derived from "objectSearch", call it "ouSearch", and set its "pattern" property to "ou":

```
<renderer id="ouSearch" extends="objectSearch">
  <renderer-property name="pattern" value="${ou}"/>
</renderer>
```

4.5.2. Reassigning Code Snippets

You can derive a new renderer from another one but assign one or more different code snippets to the new renderer.

The following renderer extends our sample renderer "customBoolean" for boolean properties:

```
<renderer id="customTrueFalse" extends="customBoolean"
       jsURL="/WEB-INF/snippets/checkbox/checkbox.js">
    <renderer-property name="labelTrue"
value="custom.boolean.true"/>
    <renderer-property name="labelFalse"
value="custom.boolean.false"/>
</renderer>
```

In HTML forms, the renderer displays boolean properties in the same way as its parent renderer, but assigns different message keys to the radio button labels. In HTML table columns, however, the renderer uses a standard code snippet that displays the properties in check boxes.

4.6. Assigning Renderers to Properties

4.6.1. Assigning a Renderer to Specific Form Fields

In form beans, you can assign the renderer to form fields or table columns, thereby overriding the default renderers for the properties displayed in the fields or columns.

4.6.1.1. Form Property

The renderer used to display or modify a property in an HTML form may be assigned as

field renderer to a form property:

```
<form-property name="dxrUserAssignmentPossible"
    type="java.lang.Boolean" fieldRenderer="customBoolean"
    readonly="false"/>
```

The above definition assigns renderer “customBoolean” to the HTML form field for property “dxrUserAssignmentPossible”.

4.6.1.2. Data Property

The renderer used to display or modify a property in an HTML table column may be assigned as cell renderer to a data property:

```
<data-property name="dxrUserAssignmentPossible"
    type="java.lang.Boolean" cellRenderer="customBoolean"
    width="5%" sortIndexes="4,0"/>
```

The above definition assigns renderer “customBoolean” to the HTML table column for property “dxrUserAssignmentPossible”.

4.6.2. Assigning a Renderer as Default Renderer

Default renderers apply to form or data properties which no specific renderer is assigned to. They apply especially to the properties in the generically generated forms used to perform request workflow tasks since these forms are not associated with any form bean.

You can associate a default renderer to a specific property, or to all properties of a specific type. For generically generated forms, you can also associate renderers with editors that are assigned to properties in object descriptions.

Assignment priority is property name first, editor second, and property type last.

Add custom assignments to file
WEB-INF/custom/config/defaultRenderer.properties.

4.6.2.1. Property Name

The following line assigns the default renderer “customBoolean” to property “dxrUserAssignmentPossible”:

```
dxruserassignmentpossible = customBoolean
```

For generic forms, the default renderer may vary with the object type:

```
dxruserassignmentpossible = customBoolean
dxruserassignmentpossible@dxrrole = checkbox
```

The default renderer for the property is “checkbox” for roles, and “customBoolean” otherwise.

Property names and object types must be specified in lower case.

4.6.2.2. Editor

The following line assigns the default renderer “customBoolean” to all properties with editor “siemens.dxm.storage.beans.JnbBoolean”:

```
siemens.dxm.storage.beans.JnbBoolean = customBoolean
```

4.6.2.3. Property Type

The following line assigns the default renderer “customBoolean” to all properties of type ”java.lang.Boolean”:

```
java.lang.Boolean = customBoolean
```

4.7. Renderer Samples

By means of two simple samples, this chapter briefly summarizes the steps necessary to write customized renderers.

The renderer **customItalicText** displays text in table cells as usual but in italics.

The renderer **customRedText** displays text in (read-only and editable) form fields and in table cells as usual but in red color.

4.7.1. Renderer Definitions

The definitions specify the code snippets to be used for the customized renderers.

4.7.1.1. WEB-INF/custom/config/renderers-config.xml

```
<!-- ========== Renderers with custom snippets ========== -->

<renderer id="customItalicText" extends="text"
      jsURL="/WEB-INF/custom/snippets/textField/italicTextField.js"/>
```

```
<renderer id="customRedText"
      defURL="/WEB-INF/custom/snippets/textField/roRedTextField.htm"
      altURL="/WEB-INF/custom/snippets/textField/rwRedTextField.htm"
      jsURL="/WEB-INF/custom/snippets/textField/redTextField.js"/>
```

4.7.2. Renderer Snippets

The snippets contain the Javascript and HTML code used for the customized renderers. The HTML snippets are copied from the snippets for the default renderer **text**; the only difference is the value of the style attribute.

4.7.2.1. WEB-INF/custom/snippets/textField/italicTextField.js

```
CUSTOM_IT("${value}")
```

4.7.2.2. WEB-INF/custom/snippets/textField/redTextField.js

```
CUSTOM_RT("${value}")
```

4.7.2.3. WEB-INF/custom/snippets/textField/roRedTextField.htm

```
<input type="text" id="${id}" name="${name}" class="roTextField"
       style="color:red;${style.h}" value="${value.h}"
       readonly="readonly" tabindex="-1"/>
```

4.7.2.4. WEB-INF/custom/snippets/textField/rwRedTextField.htm

```
<input type="text" id="${id}" name="${name}" class="rwTextField"
       style="color:red;${style.h}" value="${value.h}"
       onchange="markChanged()"/>
<script>validation.init("${id}",${mandatory});</script>
```

4.7.3. Javascript Code

The code for the Javascript functions to render values in table cells. The registration maps the renderer ids used in the above Javascript snippets to the corresponding renderer functions.

4.7.3.1. resources/custom/scripts/custom.js

```
var custom = {
    /* ==================== Custom renderers ==================== */

    italicText: function(p,n,m,v,phValue) {
        p.style.fontStyle = "italic";
        p.style.fontStretch = "expanded";
        return getTextNode(v||phValue);
    },

    redText: function(p,n,m,v,phValue) {
        p.style.color = "#F00"; // red
        return getTextNode(v||phValue);
    },
    ...
};

renderers.register(
    [ "CUSTOM_IT", "noArgs", custom.italicText ],
    [ "CUSTOM_RT", "noArgs", custom.redText ]
);
```

5. Validation

5.1. Overview

Web Center provides a built-in validation mechanism to check user input on the client side. Validation samples are:

- Checking for allowed characters, like letters or digits.
- Checking for a valid e-mail address or phone number.
- Checking the range of a number.
- Assuring that a minimum value doesn't exceed the maximum value.
- Assuring that a start date precedes the end date.
- Assuring that a date lies within a specified range.
- Assuring that an employee number is specified for internal employees.

On detection of invalid input into a form field, the field is rendered with a red background color, and form submission is blocked. Tooltips inform the user of any conditions input values must meet.

A validation method is implemented by a validator. Validators are implemented in Javascript and executed in the browser, whenever the user presses a key in an input field or changes its value. Therefore, validators should terminate quickly in order not to interrupt the user.

Validators are assigned to specific renderers, which in turn can be assigned to individual form properties, or defined as default renderer for all properties of a given name, type or editor. Note that default renderers also apply to dynamically generated forms used to perform request workflow tasks (like enter attributes activities).

Most validators check the value of a single property, while others cross-check the values of several properties, like the order of a start and end date. You can define a single-property check and a multiple-property check for the same property. You should, however, avoid conflicting checks.

Validators should not check if mandatory fields contain a value since this is already controlled by a respective attribute of form properties and request workflow activity parameters.

Property values are trimmed before being validated, that is any leading and trailing white spaces (like spaces, tabs, carriage returns and line feeds) are cut off.

While Web Center is shipped with a set of predefined validators, you may also provide custom validators.

Validation of editable table cells works a bit differently and is described in a separate chapter of this document.

The next chapters describe in detail how to write validators. To get a first feeling of how it works, take a look at the validator samples in the last chapter.

5.2. Validators

5.2.1. Single-Property Validators

A single-property validator consists of

- A validator name; the name must be a valid Javascript variable name; required.
- A list of validator arguments; optional.
- A Javascript object with two members:
  - **expr** – A regular expression that the property values must match; optional.
  - **check** – A function that checks the validity of a property value; optional. The function accepts the value as its first argument, and the validator arguments as additional arguments. The function must return “false” if the check fails, and “true” otherwise.
- Tooltip texts; per supported language; optional. Tooltips may contain placeholders to be replaced with the actual validator arguments at runtime. “${0}” is replaced with the first argument, “${1}” with the second, etc. The placeholders for date arguments are “${0d}”, “${1d}” etc.



- A custom tooltip must be assigned to

```
messages.custom.texts.tooltip.format.<validator name>
```

You can define a separate tooltip for Firefox. This is sometimes useful for tooltips including new lines, since Firefox and Internet Explorer render new lines in a different way (Firefox just ignores new lines):

```
messages.custom.texts.tooltip.format.ff.<validator
name>
```

5.2.1.1. Predefined Validators

5.2.1.1.1. fax

The validator applies to form properties of type **java.lang.String**. It checks if a field value is a facsimile telephone number complying with RFC 2252 – LDAP (v3): Attribute Syntax Definitions. The validator doesn’t accept any arguments.

5.2.1.1.2. mail

The validator applies to form properties of type **java.lang.String**. It checks (rather informally) if a field value is a valid mail address. The validator doesn’t accept any

arguments.

5.2.1.1.3. print

The validator applies to form properties of type **java.lang.String**. It checks if a field value is a valid printable string complying with RFC 2252 – LDAP (v3): Attribute Syntax Definitions. The validator doesn't accept any arguments.

5.2.1.1.4. integer

The validator applies to form properties of type **java.lang.Integer**. It checks if a field value is a whole number. Optional arguments are minimum and/or maximum value.

5.2.1.1.5. date

The validator applies to form properties of type **java.util.Date**. It checks if a day (without time) lies within a given range. Arguments are earliest and/or latest possible date. The arguments can be either specified as specific days ("yyyymmdd", e.g. "20100101" and "99991231"), or as number of days from today ("TODAY", "TODAY+10", "TODAY-365"). The year must lie within 1000 and 9999.

5.2.1.1.6. time

The validator applies to form properties of type **java.util.Date**. It checks if a date (day and time) lies within a given range. Arguments are earliest and/or latest possible date. The arguments can be either specified as specific UTC times ("yyyy-mm-ddTHH:mm:ssZ", e.g. "2000-01-01T13:00:30" and "2020-12-31T23:59Z"), or as number of days, hours or minutes from now ("NOW", "NOW+100d", "NOW-24h", "NOW-30m", where "NOW" is the time a user loads a page to enter the time). The year must lie within 1000 and 9999. Note that the terminal letter "Z" and the seconds part ":ss" in specified UTC times are optional.

Web Center users will see and enter all times in their local time, which is usually different from UTC time. This also applies to the upper and lower time bounds specified here.

5.2.1.2. Sample Custom Validators

5.2.1.2.1. customEmployeeNumber

The validator applies to form properties of type **java.lang.String**. It checks if an employee number starts with two digits, followed by a dash and two upper case letters.

- Validator name: **customEmployeeNumber**
- Validator arguments: none
- Javascript object name: **custom.employeeNumber**
- Javascript regular expression: /^\d\d-[A-Z][A-Z]$/
- Javascript function: none
- Tooltips:
- English: "Two digits (0-9), followed by a dash and two upper case letters (A-Z)."

- German: “Zwei Ziffern (0-9), gefolgt von einem Bindestrich und zwei Großbuchstaben (A-Z).“

Define the Javascript object in a custom Javascript file:

```
var custom = {
    ...
    employeeNumber: {
        expr: /^\d\d-[A-Z][A-Z]$/
    },
    ...
}
```

Add the tooltips to the object **messages.custom.texts.tooltip.format** in the custom message files (**messages.js**):

English message file:

```
customEmployeeNumber = "Two digits (0-9), followed by a dash and two upper case letters (A-Z)."
```

German message file:

```
customEmployeeNumber = "Zwei Ziffern (0-9), gefolgt von einem Bindestrich und zwei Gro\u00DFbuchstaben (A-Z)."
```

Note that non-ASCII characters must be replaced with their Unicode representations in message files, like “\u00DF” for “ß”.

5.2.1.2.2. customUid

The validator applies to form properties of type **java.lang.String**. It checks if a unique ID contains just digits and upper case letters. In addition to that, the unique ID should not exceed a configurable maximum length. We use a regular expression to ensure that a unique ID contains digits and numbers only, and a Javascript function to check the length.

- Validator name: **customUid**
- Validator arguments:
- 1st argument: The maximum length.
- Javascript object name: **custom.uid**
- Javascript regular expression: /^[\dA-Z]*$/
- Javascript function: see below

- Tooltips:
- English: “A unique ID must include digits (0-9) and upper case letters (A-Z) only. Its maximum length is ${0}.”
- German: “Eine eindeutige ID darf nur Ziffern (0-9) und Großbuchstaben (A-Z) enthalten und höchstens ${0} Zeichen lang sein).“

Define the Javascript object in a custom Javascript file:

```
var custom = {
    ...
    uid: {
        expr: /^[\dA-Z]*$/,
        check: function(value,maxLen) {
            if (!maxLen || maxLen <= 0 || !value)
                return true;
            return value.length <= maxLen;
         }
    },
    ...
}
```

Add the tooltips to the object **messages.custom.texts.tooltip.format** in the custom message files (**messages.js**):

English message file:

```
customUid = "A unique ID must include only digits (0-9) and upper
case letters (A-Z). Its maximum length is ${0}."
```

German message file:

```
customUid = "Eine UID darf nur Ziffern (0-9) und Gro\u00DFbuchstaben
(A-Z) enthalten und h\u00F6chstens ${0} Zeichen lang sein."
```

5.2.1.2.3. customPhone

The validator applies to form properties of type **java.lang.String**. It checks if a phone number complies with the format “+<country code> <area code> <local number>” where the country code is one of a list of predefined country codes, while area code and local number are non-empty sequences of digits. The country code list should be passed in as an argument to the validator.

- Validator name: **customPhone**
- Validator arguments:
  - 1st argument: The valid country codes.
- Javascript object name: **custom.phone**
- Javascript regular expression: none
- Javascript function: see below
- Tooltips:
- English: “The phone number format is \"+<Country code> <Area code> <Local number>\". \nValid country codes are: ${0}.”
- German: “Eine Telephonnummer im Format \"+<Ländercode> <Vorwahl> <Lokale Nummer>\". \nZulässige Ländercodes sind: ${0}.“

Define the Javascript object in a custom Javascript file. The list of valid country codes is expected to be a Javascript array.

```
var custom = {
    ...
    phone: {
        check: function(value,countryCodes) {
            if (!value)
                return true;
            var m = value.match(/^[+](\d+) \d+ \d+$/);
            if (m) {
                if (!countryCodes || countryCodes.length === 0)
                    return true;
                for (var i = 0; i < countryCodes.length; i++)
                    if (m[1] === countryCodes[i])
                        return true;
            }
            return false;
        }
    },
    ...
}
```

Add the tooltips to the object **messages.custom.texts.tooltip.format** in the custom message files (**messages.js**):

English message file:

```
customPhone = "The phone number format is \"+<Country code> <Area
code> <Local number>\". \nValid country codes are: ${0}."
```

German message file:

```
customPhone = "Eine Telephonnummer im Format \"+<L\u00E4ndercode>
<Vorwahl> <Lokale Nummer>\". \nZul\u00E4ssige L\u00E4ndercodes sind:
${0}."
```

5.2.1.2.4. customPrime

The validator applies to form properties of type **java.lang.Integer**. It checks if a value is a prime number between a configurable minimum and maximum value.

- Validator name: **customPrime**
- Validator arguments:
- 1st argument: The minimum value.
- 2nd argument: The maximum value.
- Javascript object name: **custom.prime**
- Javascript regular expression: /^[\d]+$/
- Javascript function: see below
- Tooltips:
- English: “A prime number from ${0} to ${1}.”
- German: “Eine Primzahl von ${0} bis ${1}.“

Define the Javascript object in a custom Javascript file:

```
var custom = {
    ...
    prime: {
        expr: /^[\d]+$/,
        check: function(value, min,max) {
            if (!value)
                return true;
            var n = Number(value);
            min = min || 2;
            if (n < 2 || n < min || (max && n > max))
                return false;
            if (n === 2)
```

```
                return true;
            if (n % 2 === 0)
                return false;
            var maxDiv = Math.floor(Math.sqrt(n));
            for (var i = 3; i < maxDiv; i += 2)
                if (n % i === 0)
                    return false;
            return true;
        }
    },
    ...
}
```

Add the tooltips to the object **messages.custom.texts.tooltip.format** in the custom message files (**messages.js**):

English message file:

```
customPrime = "A prime number from ${0} to ${1}."
```

German message file:

```
customPrime = "Eine Primzahl von ${0} bis ${1}."
```

5.2.2. Multiple-Property Validators

A multiple-property validator consists of

- A validator name; the name must be a valid Javascript variable name; required.
- A list of validator arguments; optional.
- A Javascript object with two members:
- **init** – A function to initialize the validator; optional. The function is called whenever the validator is assigned to a form property.
- **check** – A function that checks the validity of the property values. The function accepts the following arguments:
- The nodes (form elements) to be checked (a Javascript array).
- A flag indicating whether the actual invocation was triggered by an onchange or an onkeyup event.
- The optional validator arguments.

The function must return an object with two fields named “ok” and ”inv”. “inv” returns an

array of the indexes of the nodes to be added to the list of invalid properties, while “ok” returns the indexes of the nodes to be removed from that list. Any property not referenced in the arrays is left untouched.

- Tooltip texts; per supported language; optional. You can define a single tooltip for all properties, or different tooltips per property. Tooltips may contain placeholders to be replaced with the actual validator arguments at runtime. ${0} is replaced with the first argument, ${1} with the second, etc. The placeholders for date arguments are ${0d}, ${1d} etc.



- Each custom tooltip must be assigned to

```
messages.custom.texts.tooltip.format.<tooltip name>.
```

You can define separate tooltips for Firefox. This is sometimes useful for tooltips including new lines, since Firefox and Internet Explorer render new lines in a different way (Firefox just ignores new lines):

```
messages.custom.texts.tooltip.format.ff.<tooltip
name>.
```

5.2.2.1. Predefined Validators

5.2.2.1.1. minMax

The validator applies to form properties of type **java.lang.Integer**. It checks if the values of a list of number properties are in ascending order. Validator arguments are:

- The property names.
- The minimum value (**false** for no minimum).
- The maximum value (**false** for no maximum).
- Whether to check for “less than” (**true**) or “less than or equal to” (**false**, default).

  Minimum and maximum value are always checked with <= and >=, respectively.

5.2.2.1.2. dates

The validator applies to form properties of type **java.util.Date**. It checks if the values of a list of date properties (no time components) are in ascending order. Validator arguments are:

- The property names.
- Whether to check for “before” (**true**) or “before or equal to” (**false**, default).

5.2.2.1.3. times

The validator applies to form properties of type **java.util.Date**. It checks whether the values of a list of date properties (including the time components) are in ascending order. Validator arguments are:

- The property names.
- Whether to check for “before” (**true**) or “before or equal to” (**false**, default).

5.2.2.2. Sample Custom Validators

5.2.2.2.1. customEmployee

The validator checks the combination of employee type and employee number. An employee number is comprised of a configurable number of digits. The number is required for internal employees, and optional otherwise.

- Validator name: **customEmployee**
- Validator arguments:
- 1st argument: The names of properties employee type and employee number.
- 2nd argument: The number of digits for the employee number.
- Javascript object name: **custom.employee**
- Javascript init function: none
- Javascript check function: see below
- Tooltips:
- English: “An employee number is comprised of exactly ${0} digits (0-9). \nThe employee number is required for internal employees.”
- German: “Eine Mitarbeiternummer besteht aus genau ${0} Ziffern (0-9). \nF\u00FCr interne Mitarbeiter muss eine Nummer angegeben werden.“

Define the Javascript object in a custom Javascript file:

```
var custom = {
    ...
    employee: {
        check: function(nodes,onChange,numDigits) {
            if (nodes.length < 2)
                return null;
            var empType   = nodes[0];
            var empNumber = nodes[1];

            var v = trimSpaces(empNumber.value);
            onChange && (empNumber.value = v);
```

```
            var expr = new RegExp("^(?:[0-9]{"+numDigits+"})$");
            var ok   = v ? expr.test(v) : !empNumber.mandatory;

            if (ok && empType.selectedIndex !== -1) {
                if ("Internal" ===
                        empType.options[empType.selectedIndex].value)
                    ok = v.length > 0;
            }

            return ok ? {ok:[1]} : {inv:[1]};
        }
    },
    ...
}
```

The function marks the employee number property as invalid if the check fails, and as valid if the check succeeds. The employee type property is neither returned as valid nor invalid.

Add the tooltips to the object **messages.custom.texts.tooltip.format** in the custom message files (**messages.js**):

English message file:

```
customEmployee = "An employee number is comprised of exactly ${0}
digits (0-9). \nThe employee number is required for internal
employees."
```

German message file:

```
customEmployee = "Eine Mitarbeiternummer besteht aus genau ${0}
Ziffern (0-9). \nF\u00FCr interne Mitarbeiter muss eine Nummer
angegeben werden"
```

5.2.3. Registering Validators

Web Center provides a Javascript object named **validators** which controls form field validation. The object keeps a map of validator names to validator objects. Initially, the map is filled with the standard validators provided with Web Center.

Custom validators be registered with the **validators** object in order to get added to the map. The object provides a registration method:

- **register(validator ...)** – Registers one or more validators. The function accepts one or

more arguments. Each argument is an array of two items:

- The validator name.
- The validator object.

You can for example register your custom validators one by one:

```
validators.register(["customEmployee", custom.employee]);
validators.register(["customEmployeeNumber", custom.employeeNumber]);
validators.register(["customPhone", custom.phone]);
validators.register(["customUid", custom.uid]);
```

or all validators in a single step:

```
validators.register(
    [ "customEmployee", custom.employee ],
    [ "customEmployeeNumber", custom.employeeNumber ],
    [ "customPhone", custom.phone ],
    [ "customUid", custom.uid ]
);
```



- Register custom validators from within custom script files (**custom.js**).

5.3. Validation Renderers

A validation renderer uses a validator to check property values. There are a couple of base renderers to derive specific predefined and custom renderers from. As usual, each renderer applies to form properties of a specific type.

As with validators, some renderers validate the values of a single property, while others cross-check the values of multiple properties.

5.3.1. Single-Property Renderers

Single-valued properties have simple types like **java.lang.String**, **java.lang.Integer** or **java.util.Date**.

5.3.1.1. Base Renderers

5.3.1.1.1. formattedText

A base renderer for form properties of type **java.lang.String**.

```
<renderer id="formattedText" extends="text"
```

```
        altURL="/WEB-INF/snippets/textField/formattedTextField.htm"
        jsURL="/WEB-INF/snippets/textField/formattedTextField.js">
    <renderer-property name="validator" value=""/>
    <renderer-property name="args"      value=""/>
    <renderer-property name="tooltip"   value=""/>
</renderer>
```

Renderer properties:

- **validator –** Accepts the name of a specific text format validator; required.
- **args –** Accepts additional arguments for the validator; optional.
- **tooltip –** Accepts the last component of the message key for the tooltip.

5.3.1.1.2. integer

A base renderer for form properties of type **java.lang.Integer**. The renderer checks if a field value is a whole number, using the predefined validator **integer**.

```
<renderer id="integer" type="java.lang.Integer"
     defURL="/WEB-INF/snippets/textField/roInteger.htm"
        altURL="/WEB-INF/snippets/textField/formattedTextField.htm"
        jsURL="/WEB-INF/snippets/textField/formattedTextField.js">
    <renderer-property name="validator" value="integer"/>
    <renderer-property name="args"      value=""/>
    <renderer-property name="tooltip"   value=""/>
</renderer>
```

Renderer properties:

- **validator –** Accepts the name of the a number format validator; required. The name of the predefined validator is “integer”.
- **args –** Accepts additional arguments for the validator; optional. The predefined validator **integer** accepts the minimum and/or maximum value.
- **tooltip –** Accepts the last component of the message key for the tooltip.

5.3.1.1.3. boundedNumber

A base renderer for form properties of type **java.lang.Integer**. It serves to check number fields with minimum and maximum value. It just adds an appropriate tooltip to its base renderer **integer**.

```
<renderer id="boundedNumber" extends="integer">
    <renderer-property name="tooltip" value="boundedNumber"/>
```

```
</renderer>
```

5.3.1.1.4. lowerBoundedNumber

A base renderer for form properties of type **java.lang.Integer**. It serves to check number fields with minimum value. It just adds an appropriate tooltip to its base renderer **integer**.

```
<renderer id="lowerBoundedNumber" extends="integer">
    <renderer-property name="tooltip" value="lowerBoundedNumber"/>
</renderer>
```

5.3.1.1.5. calendar

The base renderer for form properties of type **java.util.Date** without the time component. It ensures that a field value is a date. It sets the validator property but leaves tooltip and arguments unspecified.

```
<renderer id="calendar" type="java.util.Date"
          defURL="/WEB-INF/snippets/textField/roTextField.htm"
          altURL="/WEB-INF/snippets/calendar/calendar.htm"
          jsURL="/WEB-INF/snippets/calendar/calendar.js">
    <renderer-property name="format"
                       value="application.dateFormat"/>
    <renderer-property name="validator" value="date"/>
    <renderer-property name="args"      value=""/>
    <renderer-property name="tooltip"   value=""/>
</renderer>
```

5.3.1.1.6. boundedDate

A base renderer for form properties of type **java.util.Date**. It is used to check if a date does not precede the earliest possible date or exceed the latest one. The renderer just adds an appropriate tooltip to its base renderer **calendar**.

```
<renderer id="boundedDate" extends="calendar">
    <renderer-property name="tooltip" value="boundedDate"/>
</renderer>
```

lowerBoundedDate

A base renderer for form properties of type **java.util.Date**. It is used to check if a date does not precede the earliest possible date. The renderer just adds an appropriate tooltip to its

base renderer **calendar**.

```
<renderer id="lowerBoundedDate" extends="calendar">
    <renderer-property name="tooltip" value="lowerBoundedDate"/>
</renderer>
```

5.3.1.1.7. upperBoundedDate

A base renderer for form properties of type **java.util.Date**. It is used to check if a date does not exceed the latest possible date. The renderer just adds an appropriate tooltip to its base renderer **calendar**.

```
<renderer id="upperBoundedDate" extends="calendar">
    <renderer-property name="tooltip" value="upperBoundedDate"/>
</renderer>
```

5.3.1.1.8. time

The base renderer for form properties of type **java.util.Date** that include a time component without seconds. It ensures that a field value is a properly formatted date with time. It sets the validator property but leaves tooltip and arguments unspecified.

```
<renderer id="time" type="java.util.Date"
              defURL="/WEB-INF/snippets/time/display.htm"
              altURL="/WEB-INF/snippets/time/edit.htm"
              jsURL="/WEB-INF/snippets/time/edit.js"
              roJsURL="/WEB-INF/snippets/time/display.js">
    <renderer-property name="format" value="ECMASCRIPT"/>
    <renderer-property name="displayFormat"
                       value="application.timeDisplayFormat"/>
    <renderer-property name="editFormat"
                       value="application.timeEditFormat"/>
    <renderer-property name="validator" value="time"/>
    <renderer-property name="args" value=""/>
    <renderer-property name="tooltip" value=""/>
    <renderer-property name="buttonTooltip"
                       value="application.tooltip.time"/>
    <!-- Activate this section for non-editable date input fields -->
    <!--
    <renderer-property name="formatTooltip" value=""/>
    <renderer-property name="formInputReadOnly"
```

```
        value="readonly=&amp;quot;readonly&amp;quot; tabindex=&amp;quot;-
1&amp;quot;"/>
    <renderer-property name="formInputStyleClass"
                       value="roTextField"/>
    <renderer-property name="tableInputStyleClass"
                       value="roTextField"/>
    <renderer-property name="tableInputReadOnly" value="true"/>
    -->
    <!-- Activate this section for editable date input fields -->
    <renderer-property name="formatTooltip"
                       value="application.timeFormat.tooltip"/>
    <renderer-property name="formInputReadOnly" value=""/>
    <renderer-property name="formInputStyleClass"
                       value="rwTextField"/>
    <renderer-property name="tableInputStyleClass"
                       value="rwTextField rwTime"/>
    <renderer-property name="tableInputReadOnly" value="false"/>
</renderer>
```

The derived renderers **boundedTime**, **lowerBoundedTime** and **upperBoundedTime** just add appropriate tooltips.

5.3.1.1.9. timeWithSeconds

The base renderer for form properties of type **java.util.Date** that include a time component with seconds. It extends renderer **time** and just redefines some formats and tooltips:

```
<renderer id="timeWitSeconds" extends="time">
    <renderer-property name="displayFormat"
                   value="application.timeWithSecondsDisplayFormat"/>
    <renderer-property name="editFormat"
                   value="application.timeWithSecondsEditFormat"/>
    <renderer-property name="formatTooltip"
                   value=
"application.timeWithSecondsFormat.tooltip"/>
</renderer>
```

The derived renderers **boundedTimeWithSeconds**, **lowerBoundedTimeWithSeconds** and **upperBoundedTimeWithSeconds** just add appropriate tooltips.

5.3.1.2. Predefined Renderers

5.3.1.2.1. fax

A renderer for form properties of type **java.lang.String**. It checks if a field value is a facsimile telephone number.

```
<renderer id="fax" extends="formattedText">
    <renderer-property name="validator" value="fax"/>
</renderer>
```

5.3.1.2.2. mail

A renderer for form properties of type **java.lang.String**. It checks (rather informally) if a field value is a valid mail address.

```
<renderer id="mail" extends="formattedText">
    <renderer-property name="validator" value="mail"/>
</renderer>
```

5.3.1.2.3. mailList

A renderer for multi-valued form properties of type **java.lang.String[]**. It checks if the field values are valid mail addresses.

```
<renderer id="mailList" extends="stringList">
    <renderer-property name="itemRenderer" value="mail"/>
</renderer>
```

5.3.1.2.4. phone

A renderer for form properties of type **java.lang.String**. It checks if a field value is a valid phone number. It uses the printable string validator since the LDAP syntax for phone numbers is printable string.

```
<renderer id="phone" extends="formattedText">
    <renderer-property name="validator" value="print"/>
</renderer>
```

5.3.1.2.5. naturalNumber

A renderer for form properties of type **java.lang.Integer**. It checks if a field value is a whole number greater than 0.

```
<renderer id="naturalNumber" extends="lowerBoundedNumber">
    <renderer-property name="args" value="1"/>
</renderer>
```

5.3.1.2.6. nonNegativeInteger

A renderer for form properties of type **java.lang.Integer**. It checks if a field value is a whole number greater than or equal to 0.

```
<renderer id="nonNegativeInteger" extends="lowerBoundedNumber">
    <renderer-property name="args" value="0"/>
</renderer>
```

5.3.1.2.7. integerList

A renderer for multi-valued form properties of type **java.lang.Integer[]**. It checks if the field values are whole numbers.

```
<renderer id="integerList" type="java.lang.Integer[]"
    className="com.siemens.webMgr.taglib.view.
                        renderers.StringListRenderer"
    defURL="/WEB-INF/snippets/textField/roStringList.htm"
    altURL="/WEB-INF/snippets/stringList/rwStringList.htm">
    <renderer-property name="hint" value="application.addValueHint"/>
    <renderer-property name="itemRenderer" value="integer"/>
</renderer>
```

5.3.1.3. Sample Custom Renderers

5.3.1.3.1. customEmployeeNumber

A renderer for form properties of type **java.lang.String**. It checks if a field value is a valid employee number, using the **customEmployeeNumber** validator.

```
<renderer id="customEmployeeNumber" extends="formattedText">
    <renderer-property name="validator" value=
"customEmployeeNumber"/>
</renderer>
```

5.3.1.3.2. customEmployeeNumberList

A renderer for multi-valued form properties of type **java.lang.String[]**. It checks if the field values are valid employee numbers.

```
<renderer id="customEmployeeNumberList" extends="stringList">
    <renderer-property name="itemRenderer"
                       value="customEmployeeNumber"/>
</renderer>
```

5.3.1.3.3. customUid

A renderer for form properties of type **java.lang.String**. It checks if a field value is a valid unique id of maximum length 8, using the **customUid** validator.

```
<renderer id="customUid" extends="formattedText">
    <renderer-property name="validator" value="customUid"/>
    <renderer-property name="args"      value="8"/>
</renderer>
```

5.3.1.3.4. customPhone

A renderer for form properties of type **java.lang.String**. It checks for phone numbers with area codes 1, 41, 43 and 49, using the **customPhone** validator.

```
<renderer id="customPhone" extends="formattedText">
    <renderer-property name="validator" value="customPhone"/>
    <renderer-property name="args"      value=
"['1','41','43','49']"/>
</renderer>
```

5.3.1.3.5. customPhoneList

A renderer for multi-valued form properties of type **java.lang.String[]**. It checks if the field values are valid phone numbers.

```
<renderer id="customPhoneList" extends="stringList">
    <renderer-property name="itemRenderer" value="customPhone"/>
</renderer>
```

5.3.1.3.6. customAge

A renderer for form properties of type **java.lang.Integer**. It checks if a field value is a whole number greater than or equal to 0 but less than or equal to 120.

```
<renderer id="customAge" extends="boundedNumber">
    <renderer-property name="args" value="0,120"/>
</renderer>
```

5.3.1.3.7. customPrime

A renderer for form properties of type **java.lang.Integer**. It checks if a field value is a prime number less than or equal to 10000.

```
<renderer id="customPrime" extends="integer">
    <renderer-property name="validator" value="customPrime"/>
    <renderer-property name="args"      value="10000"/>
</renderer>
```

5.3.1.3.8. customPrimeList

A renderer for multi-valued form properties of type **java.lang.Integer[]**. It checks if each field value is a prime number less than or equal to 10000.

```
<renderer id="customPrimeList" extends="integerList">
    <renderer-property name="itemRenderer" value="customPrime"/>
</renderer>
```

5.3.1.3.9. custom21stCentury

A renderer for form properties of type **java.util.Date**. It checks if a date lies within the 21st century.

```
<renderer id="custom21stCentury" extends="boundedDate">
    <renderer-property name="args" value="'20000101','20991231'"/>
</renderer>
```

5.3.1.3.10. customOneYear

A renderer for form properties of type **java.util.Date**. It checks if a date lies within one year from today.

```
<renderer id="customOneYear" extends="boundedDate">
    <renderer-property name="args" value="'TODAY','TODAY+365'"/>
</renderer>
```

5.3.2. Multiple-Property Renderers

Multi-valued properties have array types like **java.lang.String[]** or **java.lang.Integer[]**.

To obtain a validation renderer for a multi-valued property, extend a suitable predefined non-validating renderer for the type of the property and assign a suitable validation renderer as item renderer. Note that it is not possible to pass individual arguments to the item renderer.

5.3.2.1. General Renderer Properties

5.3.2.1.1. ids

The names of the form properties to be checked by the renderer. A comma-separated list. Samples are

- **dxrStartDate,dxrDisableStartDate,dxrDisableEndDate,dxrEnddate**
- **dxrMinPwdLength,dxrMaxPwdLength**

5.3.2.1.2. tooltips

The tooltips to be displayed for the form properties. You may define a single tooltip for all properties, or individual tooltips per property, and even skip tooltips for some properties. Note that the tooltips are appended to any other tooltip defined for a property. Specify a tooltip with its message id relative to “messages.custom.texts.tooltip.format” or “messages.texts.tooltip.format” resp.

Samples are:

- **“all”** – The same tooltip for all properties.
- **“first,secondAndHigher”** – The first tooltip applies to the first property, the second to any subsequent property.
- **“„thirdAndHigher”** – The same tooltip for the third and any subsequent property, but none for the first two properties.
- **“first„third,”** – Tooltips for the first and third property, no tooltip for the other ones.

Nth property refers to the nth property listed in the above **ids**.

5.3.2.2. Base Renderers

5.3.2.2.1. minMax

The renderer applies to a list of form properties of type **java.lang. Integer**. It checks if the

values of the number properties are in ascending order (< or <=) and lie within a given range.

```
<renderer id="minMax" extends="integer"
          altURL="/WEB-
INF/snippets/textField/formattedTextFields.htm">
    <renderer-property name="ids"        value=""/>
    <renderer-property name="mValidator" value="minMax"/>
    <renderer-property name="mArgs"      value=""/>
       <renderer-property name="tooltips"   value=""/>
</renderer>
```

Renderer properties:

- **ids –** The integer property names.
- **mValidator –** The validator name.
- **mArgs –** Arguments for the **minMax** validator; optional. Samples are:
  - **0 –** Minimum 0, no maximum, <=.
  - **1,1000 –** Minimum 0, maximum 1000, <=.
  - **false,100,true –** No minimum, maximum 100, <.
  - **10,false,true –** Minimum 10, no maximum, <.
- **tooltips –** The tooltips for the properties.



- If the renderer is applied to a form which misses some of the listed properties, the missing properties are ignored.

5.3.2.2.2. boundedMinMax

The renderer extends **minMax** by defining an appropriate tooltip for <= checks with minimum and maximum.

```
<renderer id="boundedMinMax" extends="minMax">
    <renderer-property name="tooltips" value="boundedMinMax"/>
</renderer>
```

5.3.2.2.3. boundedMinMaxLess

The renderer extends **minMax** by defining an appropriate tooltip for < checks with minimum and maximum.

```
<renderer id="boundedMinMaxLess" extends="minMax">
```

```
    <renderer-property name="tooltips" value="boundedMinMaxLess"/>
</renderer>
```

5.3.2.2.4. lowerBoundedMinMax

The renderer extends **minMax** by defining an appropriate tooltip for <= checks with minimum but without maximum.

```
<renderer id="lowerBoundedMinMax" extends="minMax">
    <renderer-property name="tooltips" value="lowerBoundedMinMax"/>
</renderer>
```

5.3.2.2.5. lowerBoundedMinMaxLess

The renderer extends **minMax** by defining an appropriate tooltip for < checks with minimum but without maximum.

```
<renderer id="lowerBoundedMinMaxLess" extends="minMax">
    <renderer-property name="tooltips"
                       value="lowerBoundedMinMaxLess"/>
</renderer>
```

5.3.2.2.6. checkedDate

The renderer applies to a list of form properties of type **java.util.Date** without time components. It checks if the values of a list of date properties are in ascending order (< or <=).

The renderer is defined as follows:

```
<renderer id="checkedDate" extends="calendar"
          altURL="/WEB-INF/snippets/calendar/checkedDate.htm">
    <renderer-property name="ids"        value=""/>
    <renderer-property name="mValidator" value="dates"/>
    <renderer-property name="mArgs"      value=""/>
    <renderer-property name="tooltips"   value=""/>
</renderer>
```

Renderer properties:

- **ids** – The date property names.
- **mValidator** – The validator name.

- **mArgs –** Arguments for the **dates** validator; optional. Samples are:
  - **true –** Use <.
  - **false –** Use <= (default).
- **tooltips –** The tooltips for the properties.



- If the renderer is applied to a form which misses some of the listed properties, the missing properties are ignored.

5.3.2.2.7. checkedTime

The renderer applies to a list of form properties of type **java.util.Date** with time components. It checks whether the values of a list of date properties are in ascending order (< or <=).

The renderer is defined as follows:

```
<renderer id="checkedTime" extends="time"
          altURL="/WEB-INF/snippets/time/checkedTime.htm">
    <renderer-property name="ids"        value=""/>
    <renderer-property name="mValidator" value="times"/>
    <renderer-property name="mArgs"      value=""/>
    <renderer-property name="tooltips"   value=""/>
</renderer>
```

Renderer properties:

- **ids –** The date property names.
- **mValidator –** The validator name.
- **mArgs –** Arguments for the **times** validator; optional. Samples are:
  - **true –** Use <.
  - **false –** Use <= (default).
- **tooltips –** The tooltips for the properties.



- If the renderer is applied to a form which misses some of the listed properties, the missing properties are ignored.
- There's a corresponding renderer **checkedTimeWithSeconds** for times including seconds.

5.3.2.3. Predefined Renderers

5.3.2.3.1. startEndDate

The renderer checks if the values of properties **dxrStartDate** and **dxrEndDate** are

consistent, ie

```
dxrStartDate <= dxrEndDate
```

The renderer is defined as follows:

```
<renderer id="startEndDate" extends="checkDate">
    <renderer-property name="ids"  value="dxrStartDate,dxrEndDate"/>
    <renderer-property name="tooltips" value="startEndDate"/>
</renderer>
```

5.3.2.3.2. startEndDateLess

The renderer checks if the values of properties **dxrStartDate** and **dxrEndDate** are consistent, ie

```
dxrStartDate < dxrEndDate
```

The renderer is defined as follows:

```
<renderer id="startEndDateLess" extends="checkDate">
    <renderer-property name="ids"   value="dxrStartDate,dxrEndDate"/>
    <renderer-property name="mArgs" value="true"/>
    <renderer-property name="tooltips" value="startEndDateLess"/>
</renderer>
```

5.3.2.3.3. startEndDateWithDisabled

The renderer checks if the values of properties **dxrStartDate**, **dxrEndDate**, **dxrDisableStartDate** and **dxrDisableEndDate** are consistent, ie

```
dxrStartDate <= dxrDisableStartDate <= dxrDisableEndDate <=
dxrEndDate
```

If a form does not include disable dates, the check reduces to

```
dxrStartDate <= dxrEndDate.
```

The renderer is defined as follows:

```
<renderer id="startEndDateWithDisabled" extends="checkdDate">
    <renderer-property name="ids"
              value="dxrStartDate,dxrDisableStartDate,
                                dxrDisableEndDate,dxrEndDate"/>
    <renderer-property name="tooltips"
              value="startEndDateWithDisabled"/>
</renderer>
```

5.3.2.3.4. startEndDateWithDisabledLess

The renderer checks if the values of properties **dxrStartDate**, **dxrEndDate**, **dxrDisableStartDate** and **dxrDisableEndDate** are consistent, ie

```
dxrStartDate < dxrDisableStartDate < dxrDisableEndDate < dxrEndDate
```

If a form does not include disable dates, the check reduces to

```
dxrStartDate < dxrEndDate
```

The renderer is defined as follows:

```
<renderer id="startEndDateWithDisabledLess" extends="checkedDate">
    <renderer-property name="ids"
              value="dxrStartDate,dxrDisableStartDate,
                                dxrDisableEndDate,dxrEndDate"/>
    <renderer-property name="mArgs" value="true"/>
    <renderer-property name="tooltips"
              value="startEndDateWithDisabledLess"/>
</renderer>
```

5.3.2.4. Sample Custom Renderers

5.3.2.4.1. customPasswordLength

The renderer checks if the required minimum password length is less than or equal to its maximum length. The minimum length must not be less than 1. Maximum value 0 means unbounded (since its less than the minimum).

```
<renderer id="customPasswordLength" extends="minMax">
    <renderer-property name="ids"
```

```
                        value="dxrPwdMinLength,dxrPwdMaxLength"/>
    <renderer-property name="mArgs" value="1,0"/>
</renderer>
```

5.3.2.4.2. customDepartureArrival

The renderer checks if departure date and arrival date are in proper order.

```
<renderer id="customDepartureArrival" extends="checkedDate">
    <renderer-property name="ids" value="departureDate,arrivalDate"/>
    <renderer-property name="tooltips" value=
"customDepartureArrival"/>
</renderer>
```

5.3.2.4.3. customEmployee

The renderer cross-checks employee type and employee number. It checks for 5-digits employee numbers. There's no tooltip for employee type, while the tooltip for employee number is "customEmployee".

```
<renderer id="customEmployee" extends="text"
       altURL="/WEB-INF/snippets/textField/formattedTextFields.htm">
    <renderer-property name="ids"
                       value="employeeType,employeeNumber"/>
    <renderer-property name="mValidator" value="customEmployee"/>
    <renderer-property name="mArgs"      value="5"/>
    <renderer-property name="tooltips"   value=",customEmployee"/>
    <renderer-property name="validator"  value=""/>
    <renderer-property name="args"       value=""/>
</renderer>
```



T0 he renderer assigns empty values to properties "validator" and "args" to avoid Javascript errors caused by the expressions "${validator.jsd}" and "${args}" used in snippet formattedTextFields.htm.

5.3.2.4.4. customStartEndTime

The renderer checks whether a start time truly precedes the end time. It defines the names of the affected properties, the tooltip to be displayed for both properties, and that the start time must not be equal to the end time (by setting the single argument to "true").

```
<renderer id="customStartEndTime" extends="checkTime">
```

```
    <renderer-property name="ids"      value="startTime,endTime"/>
    <renderer-property name="mArgs"    value="true"/>
    <renderer-property name="tooltips" value="customStartEndTime"/>
</renderer>
```

5.4. Assigning Renderers to Properties

This section gives some samples for how to assign renderers to form properties. You can assign a renderer to an individual form property in a form bean definition. On the other hand, you can define a renderer as default renderer for specific properties.

5.4.1. Assigning Renderers to Individual Form Fields

In form beans, you can assign the renderer to individual form fields, thereby overriding the default renderers.

Note that you cannot assign individual renderers to properties dynamically inserted into forms to perform request workflow tasks.

Note that validation renderers do not work with editable table cells (data properties). Do not assign validation renderers to data properties.

5.4.1.1. Single-Property Renderers

```
<!– Fax number with predefined renderer -->
<form-property name="facsimileTelephoneNumber"
type="java.lang.String"
               width="100%" y="+1"
               fieldRenderer="fax"/>
<!– Employee number with custom renderer -->
<form-property name="employeeNumber" type="java.lang.String"
               width="100%" y="+1"
               fieldRenderer="customEmployeeNumber"/>
<!– UID property with default maximum length 8 -->
<form-property name="uid" type="java.lang.String"
               width="100%" y="+1" mandatory="true"
               fieldRenderer="customUid"/>
<!– UID property with individual maximum length 10 -->
<form-property name="uid" type="java.lang.String"
               width="100%" y="+1"
               fieldRenderer="customUid"
               rendererProperties="args:10"/>
<!– Phone number property with default country codes -->
```

```
<form-property name="telephoneNumber" type="java.lang.String"
               width="100%" y="+1"
               fieldRenderer="customPhone"/>

<!– Phone number property with individual country codes -->
<form-property name="telephoneNumber" type="java.lang.String"
               width="100%" y="+1"
               fieldRenderer="customPhone"
               rendererProperties="args:['30','31','33','39']"/>
<!– Mail addresses with predefined renderer -->
<form-property name="mail" type="java.lang.String[]"
               spanX="4" spanY="3" height="65" width="100%" x="1"
               fieldRenderer="mailList"/>
<!– Phone numbers with custom renderer -->
<form-property name="telephoneNumber" type="java.lang.String[]"
               spanX="4" spanY="3" height="65" width="100%" y="+1"
               fieldRenderer="customPhoneList"/>
<!– Maximum password age with predefined renderer -->
<form-property name="dxrPwdMaxAge" type="java.lang.Integer"
               label="dxrpwdmaxage" width="50" y="+1"
               fieldRenderer="nonNegativeInteger"/>
<!– Minimum password length with predefined renderer -->
<form-property name="dxrPwdMinLength" type="java.lang.Integer"
               label="dxrpwdminlength" width="50" y="+1"
               fieldRenderer="naturalNumber"/>
<!– Two equivalent definitions: -->
<!– Age property with custom renderer -->
<!– Age property with predefined renderer and individual args -->
<form-property name="age" type="java.lang.Integer"
               width="50" y="+1"
               fieldRenderer="customAge"/>
<form-property name="age" type="java.lang.Integer"
               width="50" y="+1"
               fieldRenderer="integer"
               rendererProperties="args:0,120"/>
<!– Number list with custom renderer -->
<form-property name="primes" type="java.lang.Integer[]"
               height="65" width="200" y="+1"
               fieldRenderer="customPrimeList"/>
```

5.4.1.2. Multiple-Property Renderers

```
<!– Two equivalent definitions: -->
<!– Minimum/maximum password length with custom renderer -->
<!– Minimum/maximum password length with predefined renderer and
individual ids -->
<!– Checks for 1 <= dxrPwdMinLength <= dxrPwdMaxLength -->
<form-property name="dxrPwdMaxLength" type="java.lang.Integer"
               label="dxrpwdmaxlength" width="50"
               fieldRenderer="customPasswordLength"/>
<form-property name="dxrPwdMaxLength" type="java.lang.Integer"
               label="dxrpwdmaxlength" width="50"
               fieldRenderer="minMax"
               rendererProperties=
                    "ids:dxrPwdMinLength,dxrPwdMaxLength;mArgs:1,0"/>
<!– Date properties with predefined renderer -->
<form-property name="dxrDisableEndDate" type="java.util.Date"
               width="100" x="1"
               fieldRenderer="checkedDate"/>
<!– Departure/arrival dates with custom renderer -->
<form-property name="arrivalDate" type="java.util.Date"
               width="100" y="+1"
               fieldRenderer="customDepartureArrival"/>
<!– First/second/third dates with predefined renderer and individual
ids -->
<form-property name="third" type="java.util.Date"
               width="100" x="2"
               fieldRenderer="checkedDate"
               rendererProperties="ids:first,second,third"/>
```

5.4.1.3. Combining Single-Property and Multiple-Property Renderers

The next sample combines the single-property renderer **boundedDate** with the multiple-property renderer **startEndDateLess**. The single-property validator checks if the dates lie within the 21st century. The multiple-property validator makes sure that the start date precedes the end date.

```
<form-property name="dxrStartDate" type="java.util.Date"
               width="100" y="+1"
               fieldRenderer="boundedDate"
               rendererProperties="args:'20000101','20991231'"/>
```

```
<form-property name="dxrEndDate" type="java.util.Date"
               width="100" x="1"
               fieldRenderer="startEndDateLess"
               rendererProperties="args:'20000101','20991231';
                                   tooltip:boundedDate"/>
```

For both fields, the tooltip is the concatenation of the “boundedDate” tooltip with the “startEndDateLess” tooltip.

5.4.2. Assigning Validation Renderers as Default Renderers

Default renderers apply to static form properties defined in forms-config.xml files, as well as to dynamically generated properties of forms used to perform request workflow tasks.

File: **WEB-INF/config/defaultRenderer.properties**

```
# Predefined renderers
dxrdisableenddate         = startEndDateWithDisabled
dxrenddate                = startEndDateWithDisabled
facsimiletelephonenumber  = fax
mail                      = mailList
mobile                    = phone
telephonenumber           = phone

# Custom renderers
age                       = customAge
arrivaldate               = customDepartureArrival
dxrpwdmaxlength           = customPasswordLength
telephonenumber@dxruser   = customPhone
uid                       = customUid
```

5.5. Customization Hints

See the chapter on “Customization Files” for details.

5.5.1. Javascript Code

Add custom validators and the calls to register them to a custom Javascript file like **resources/custom/scripts/custom.js**.

Add an include instruction for the file to the custom ScriptsAndStyles-JSP:

```
<script src="resources/custom/scripts/custom.js"></script>
```

5.5.2. Tooltips

Add custom tooltips to the object messages.custom.texts in your custom message files like **resources/custom/scripts/messages/<language>/messages.js**.

Add an include instruction for the file to the custom ScriptsAndStyles-JSP:

```
<script src="resources/custom/messages/
        <c:out value="
${sessionScope['com.siemens.webMgr.language']}"/>
                      /messages.js">
</script>
```

5.5.3. Renderers

Add custom renderers to a custom renderers configuration file like **WEB-INF/custom/config/renderers-config.xml**.

5.6. Validator Samples

This section shows the implementation of some sample validators.

The validator **employeeNumber** checks:

- If an employee number matches a given regular expression.

The validator **phone** checks:

- If a phone number matches a given regular expression.
- If its country code is contained in a list of predefined country codes.

The validator **uid** checks:

- If a user id matches a given regular expression.
- If it does not exceed a given maximum length.

The validator **employee** checks a combination of employee number and employee type:

- Whether the employee number is a sequence of digits of a given length (if not empty).
- Whether the employee number is specified in case the field is mandatory.
- Whether the employee number is specified in case employee type is “Internal”.

5.6.1. Javascript Code

Each validator object defines a regular expression and/or implements a check function.

The registration defines names for the validators to be used by renderers.

5.6.1.1. resources/custom/scripts/custom.js

```
var custom = {
    ...
    /* ================== Custom validators ================== */

    /* ---------- Single-property validators ---------- */

    employeeNumber: {
        expr: /^\d\d-[A-Z][A-Z]$/
    },

    phone: {
        check: function(value,countryCodes) {
            if (!value)
                return true;
            var m = value.match(/^[+](\d+) \d+ \d+$/);
            if (m) {
                if (!countryCodes || countryCodes.length === 0)
                    return true;
                for (var i = 0; i < countryCodes.length; i++)
                    if (m[1] === countryCodes[i])
                        return true;
            }
            return false;
        }
    },

    uid: {
        expr: /^[\dA-Z]*$/,

        check: function(value,maxLen) {
            if (!maxLen || maxLen <= 0 || !value)
                return true;
            return value.length <= maxLen;
        }
    },
```

```
    /* ---------- Multiple-property validators ---------- */

    employee: {
        check: function(nodes,onChange,numDigits) {
            if (nodes.length < 2)
                return null;
            var empType   = nodes[0];
            var empNumber = nodes[1];

            var v = trimSpaces(empNumber.value);
            onChange && (empNumber.value = v);

            var expr = new RegExp("^(?:[0-9]{"+numDigits+"})$");
            var ok   = v ? expr.test(v) : !empNumber.mandatory;

            if (ok && empType.selectedIndex !== -1) {
                if ("Internal" ===
                        empType.options[empType.selectedIndex].value)
                    ok = v.length > 0;
            }

            return ok ? {ok:[1]} : {inv:[1]};
        }
    },
    ...
};

validators.register(
    [ "customEmployee", custom.employee ],
    [ "customEmployeeNumber", custom.employeeNumber ],
    [ "customPhone", custom.phone ],
    [ "customUid", custom.uid ]
);
```

5.6.2. Tooltips

5.6.2.1. resources/custom/scripts/messages/en/messages.js

```
messages.custom = {
    texts: {
```

```
        tooltip: {
            format: {
                customEmployee:       "An employee number is ...",
                customEmployeeNumber: "Two digits (0-9), ...",
                customPhone:          "The phone number format ...",
                customUid:            "A unique ID must include ..."
            }
        }
    }
};
```

6. Photos and Certificates

Web Center supports photos and certificates.

Web Center is delivered with three pre-defined renderers for photos:

- **photo** – Displays a photo.
- **photoDownload** – Displays an icon to display a photo.
- **photos** – Displays one or more photos.
- **documents** – Displays one or more photos.

Web Center is delivered with two pre-defined renderers for certificates:

- **certificateDownload** – Displays a download icon for a certificate.
- **certificate** – Displays one or more certificates, along with icons to download them.

Downloads are, in fact, supported for any binary attribute, not just photos and certificates, though there are no pre-defined renderers for other attributes.

The current version does not include pre-defined renderers for integrating binary attributes in entry lists.

6.1. Prerequisites

A binary attribute must be appropriately defined in a DirX Identity object description before you can access its values from within Web Center:

- Its type must be set to byte array: **type="[b"**.
- In order to access multiple values, its multi value flag must be set to true: **multivalue="true"**.

The following sample defines two single valued binary attributes (userCertificate and jpegPhoto), and a multivalued one (photo):

```
<property name="userCertificate" label="Certificate"
          type="[B" multivalue="false" editor="..."/>

<property name="photo" label="Photo"
          type="[B" multivalue="true" editor="..."/>

<property name="jpegPhoto" label="Photo"
          type="[B" editor="..."/>
```

6.2. Renderers

This section provides information about the pre-defined renderers for photos and certificates.

6.2.1. Photo

The renderer downloads a photo immediately when the HTML page is loaded, and displays a small thumbnail of it. When the user moves the mouse over the thumbnail, the photo is displayed in its actual size on top of the page. When the user moves the mouse off of the actual-sized photo, it is hidden again.

The renderer cannot handle more than one photo; if several photos are available, the first one is rendered.

6.2.1.1. Form Field Configuration Sample

```
<form-property name="jpegPhoto" type="byte[]"
               label="ldap.attribute.photo"
               readonly="true"
               fieldRenderer="photo"/>
```

Change the type to “java.lang.Object[]” if the property is defined as multi-valued in the DirX Identity object descriptions.

6.2.1.2. Renderer Configuration

The following sections provide information about how to configure the renderer for photos.

6.2.1.2.1. Attributes

The following table lists the attributes for renderer configuration:

Name	Value
Id	Photo
type	java.lang.Object[]
className	com.siemens.webMgr.taglib.view.renderers.DownloadRenderer
defURL	/WEB-INF/snippets/binary/photo.htm

6.2.1.2.2. Properties

The renderer does not support any properties in addition to the ones described in the section “DownloadRenderer” in the section “Java Renderer Classes”.

6.2.1.2.3. Sample

```
<renderer id="photo" type="java.lang.Object[]"
    className=
  "com.siemens.webMgr.taglib.view.renderers.DownloadRenderer"
    defURL="/WEB-INF/snippets/binary/photo.htm">
  <renderer-property name="servletPath"
                     value="binaryReader"/>
  <renderer-property name="contentType" value="image/jpeg"/>
  <renderer-property name="prefix"      value="jpegPhoto"/>
  <renderer-property name="extension"   value=".jpg"/>
</renderer>
```

The generated download URI is “binaryReader/jpegPhoto_0.jpg?...”.

6.2.1.3. HTML/Javascript Rendering

The HTML code snippet is defined in binary/photo.htm. It invokes a JavaScript function of the photos variable in documentFuncs.js.

6.2.2. PhotoDownload

The renderer displays just a small icon when the HTML page is loaded. When the user clicks the icon, the photo is downloaded from the server and displayed in its actual size on top of the page. Moving the mouse off of the photo hides it again.

The renderer cannot handle more than one photo; if several photos are available, just the first one is rendered. The renderer saves resources since the photo is downloaded only on specific request.

6.2.2.1. Form Field Configuration Sample

```
<form-property name="photo" type="byte[]"
               label="ldap.attribute.photo" readonly="true"
               fieldRenderer="photoDownload"/>
```

Change the type to “java.lang.Object[]” if the property is defined as multi-valued in the DirX Identity object descriptions.

6.2.2.2. Renderer Configuration

The following sections provide information on how to configure the renderer for photo download displaying an icon first.

6.2.2.2.1. Attributes

The following table lists the attributes for renderer configuration for photo download:

Name	Value
id	photoDownload
type	java.lang.Object[]
className	com.siemens.webMgr.taglib.view.renderers.DownloadRenderer
defURL	/WEB-INF/snippets/binary/photoDownload.htm

6.2.2.2.2. Properties

The renderer supports the following properties in addition to the properties described in the section “DownloadRenderer” of section “Java Renderer Classes”:

Name	Value
downloadImage	The icon to be displayed when the HTML page is loaded.
downloadText	The message key for the icon tooltip.

6.2.2.2.3. Sample

```
<renderer id="photoDownload" type="java.lang.Object[]"
    className=
  "com.siemens.webMgr.taglib.view.renderers.DownloadRenderer"
    defURL="/WEB-INF/snippets/binary/photoDownload.htm">
  <renderer-property name="servletPath"
                     value="binaryReader"/>
  <renderer-property name="contentType" value="image/gif"/>
  <renderer-property name="prefix"      value="photo"/>
  <renderer-property name="extension"   value=".gif"/>
  <renderer-property name="downloadImage"
                     value="resources/images/photo.gif"/>
  <renderer-property name="downloadText"
                     value="photo.display"/>
</renderer>
```

The generated download URI is “binaryReader/photo_0.gif?...”.

6.2.2.3. HTML/Javascript Rendering

The HTML code snippet is defined in binary/photoDownload.htm. It invokes a Javascript function of the photos variable in documentFuncs.js.

6.2.3. Photos

The renderer downloads and displays one or more photos when the HTML page is loaded. The photos are either displayed in their original size, or as thumbnails. A user can view a thumbnail photo in its original size by clicking on it or by moving the mouse onto it. The photos can be aligned vertically or horizontally.

6.2.3.1. Form Field Configuration Sample

```
<form-property name="photo" type="byte[]" y="+1" spanX="6"
               label="ldap.attribute.photo" readonly="true"
               fieldRenderer="photos"/>
```

Change the type to “java.lang.Object[]” if the property is defined as multi-valued in the DirX Identity object descriptions.

6.2.3.2. Renderer Configuration

The following sections provide information on how to configure the renderer for photo download displaying the photo in its actual size.

6.2.3.2.1. Attributes

The following table lists the attributes for renderer configuration:

Name	Value
id	Photos
type	java.lang.Object[]
className	com.siemens.webMgr.taglib.view.renderers.DownloadRenderer
defURL	/WEB-INF/snippets/binary/photos.htm

6.2.3.2.2. Properties

The renderer supports the following properties in addition to the ones described in the section “DownloadRenderer” of the section “Java Renderer Classes”:

Name	Value
direction	Whether to display the photos one below each other (vertical) or in a single row (horizontal).
thumbNailClass	The CSS classes for the photos. Can be used to define the display width and height of a thumbnail, or the gap between adjacent thumbnails.
event	Whether to display a photo in its original size when a user clicks onto its thumbnail (click) or moves the mouse onto it (mouseover).
hint	A message key indicating to the user how to enlarge a thumbnail.

6.2.3.2.3. Sample

```
<renderer id="photos" type="java.lang.Object[]"
    className=
  "com.siemens.webMgr.taglib.view.renderers.DownloadRenderer"
    defURL="/WEB-INF/snippets/binary/photos.htm">
  <renderer-property name="servletPath"
                     value="binaryReader"/>
  <renderer-property name="contentType" value="image/gif"/>
  <renderer-property name="prefix"      value="photo"/>
  <renderer-property name="extension"   value=".gif"/>
  <renderer-property name="max"         value="0"/>
  <renderer-property name="direction"
                     value="horizontal"/>
  <renderer-property name="thumbNailClass"
              value="photoThumbnail thumbnailHori"/>
  <renderer-property name="event"
                     value="mouseover"/>
  <renderer-property name="hint"
                     value="photo.enlarge.mouseover"/>
</renderer>
```

The generated download URIs are “binaryReader/photo_0.gif?...”, “binaryReader/photo_1.gif?...” and so on.

6.2.3.3. HTML/Javascript Rendering

The HTML code snippet is defined in binary/photos.htm. It invokes a JavaScript function of the photos variable in documentFuncs.js.

6.2.4. Documents

The renderer is a variant of renderer Photos with different values for some renderer properties.

6.2.4.1. Sample

```
<renderer id="documents" extends="photos">
  <renderer-property name="direction"
                     value="vertical"/>
  <renderer-property name="thumbNailClass"
              value="documentThumbnail thumbnailVert"/>
  <renderer-property name="event"
```

```
                    value="click"/>
  <renderer-property name="hint"
                    value="photo.enlarge.click"/>
</renderer>
```

6.2.5. Certificate Download

The renderer displays a small icon when the HTML page is loaded. When the user clicks the icon, the certificate is downloaded from the server and the browser opens its standard dialog to let the user view the certificate or save it as a file on the client.

The renderer cannot handle more than one certificate; if several certificates are available, the first one is rendered.

6.2.5.1. Form Field Configuration Sample

```
<form-property name="userCertificate" type="byte[]"
               label="ldap.attribute.certificate"
               readonly="true"
               fieldRenderer="certificateDownload"/>
```

Change the type to “java.lang.Object[]” if the property is defined as multi-valued in the DirX Identity object descriptions.

6.2.5.2. Renderer Configuration

The following sections provide information on how to configure the renderer for certificate download displaying a small icon first.

6.2.5.2.1. Attributes

The following table lists the attributes for renderer configuration:

Name	Value
id	certificateDownload
type	java.lang.Object[]
className	com.siemens.webMgr.taglib.view.renderers.DownloadRenderer
defURL	/WEB-INF/snippets/binary/certificateDownload.htm

6.2.5.2.2. Properties

The renderer supports the following properties in addition to the properties described in the section “DownloadRenderer” of the section “Java Renderer Classes”:

Name	Value
downloadImage	The icon to be displayed when the HTML page is loaded.
downloadText	The message key for the icon tooltip.

6.2.5.2.3. Sample

```
<renderer id="certificateDownload" type="java.lang.Object[]"
    className=
  "com.siemens.webMgr.taglib.view.renderers.DownloadRenderer"
    defURL=
        "/WEB-INF/snippets/binary/certificateDownload.htm">
  <renderer-property name="servletPath"
                     value="binaryReader"/>
  <renderer-property name="contentType"
                     value="application/pkix-cert"/>
  <renderer-property name="prefix"      value="certificate"/>
  <renderer-property name="extension"   value=".cer"/>
  <renderer-property name="downloadImage"
                     value="resources/images/cert.gif"/>
  <renderer-property name="downloadText"
                     value="certificate.displayDownload"/>
</renderer>
```

The resulting download URI is “binaryReader/cert_0.cer?...”.

6.2.5.3. HTML/JavaScript Rendering

The HTML code snippet is defined in binary/certificateDownload.htm. It invokes a JavaScript function of the certs variable in documentFuncs.js.

6.2.6. Certificates

The renderer displays one or more certificates, one below the other, when the HTML page is loaded. Each certificate is displayed in a table listing a configurable subset of its fields like issuer name, subject name and validity period. A small icon allows for downloading the certificate from the server; when clicked, the browser opens its standard dialog to let the user view the certificate or save it as a file on the client.

The renderer is quite simple since it displays each certificate field on a single line and truncates values exceeding a configurable maximum length.

6.2.6.1. Form Field Configuration Sample

```
<form-property name="userCertificate" type="byte[]"
               y="+1" spanX="6"
               label="ldap.attribute.certificate"
               readonly="true"
               fieldRenderer="certificates"/>
```

Change the type to “java.lang.Object[]” if the property is defined as multi-valued in the DirX Identity object descriptions.

6.2.6.2. Renderer Configuration

The following sections provide information on how to configure the renderer for certificate download displaying the certificates.

6.2.6.2.1. Attributes

The following table lists the attributes for renderer configuration:

Name	Value
id	certificates
type	java.lang.Object[]
className	com.siemens.webMgr.taglib.view.renderers.X509CertificateRenderer
defURL	/WEB-INF/snippets/binary/certificates.htm

6.2.6.2.2. Properties

In addition to the properties described in section “DownloadRenderer” of section “Java Renderer Classes” below the renderer supports the following properties:

Name	Value
downloadImage	The icon to be displayed when the HTML page is loaded.
downloadText	The message key for the icon tooltip.
maxValueLen	The maximum length for values displayed in the table. Longer values are truncated, which is indicated by three dots appended to the truncated value.

6.2.6.2.3. Sample

```
<renderer id="certificates" type="java.lang.Object[]"
    className=
       "com.siemens.webMgr.taglib.view.renderers.
                                  X509CertificateRenderer"
    defURL="/WEB-INF/snippets/binary/certificates.htm">
```

```
  <renderer-property name="dateFormat"
                     value="certificate.dateFormat"/>
  <renderer-property name="dnFormat"      value="RFC1779"/>
  <renderer-property name="attributes"
                     value="subjectDN,valid"/>
  <renderer-property name="messagePrefix"
                     value="certificate"/>
  <renderer-property name="servletPath"
                     value="binaryReader"/>
  <renderer-property name="contentType"
                     value="application/pkix-cert"/>
  <renderer-property name="prefix"
                     value="certificate"/>
  <renderer-property name="extension"     value=".cer"/>
  <renderer-property name="max"           value="0"/>
  <renderer-property name="maxValueLen"   value="80"/>
  <renderer-property name="downloadImage"
                     value="resources/images/cert.gif"/>
  <renderer-property name="downloadText"
                     value="certificate.download"/>
</renderer>
```

The generated download URIs are “binaryReader/cert_0.cer?...”, “binaryReader/cert_1.cer?...” etc.

6.2.6.3. HTML/JavaScript Rendering

The HTML code snippet is defined in binary/certificates.htm. It invokes a JavaScript function of the certs variable in documentFuncs.js.

6.3. Java Renderer Classes

This section provides information about Java Renderer classes.

6.3.1. DownloadRenderer

The renderer generates a JavaScript representation of URIs to download a user’s photos or certificates (or any other binary attribute of any other entry). It sets the following property that can be evaluated by JavaScript code in a corresponding HTML renderer snippet:

- **value** - A JavaScript array with the download URIs for the binary attribute values.

Each download URI includes:

- The servlet path of the binary reader servlet.

- Extra path information composed of a prefix, a dash, the value's index and an extension. It serves as a proposal for the file name when the user saves the binary attribute to a file on the client.
- Request parameters:
- The entry DN.
- The name of the binary attribute.
- The index of the attribute value, starting with 0.
- The value for the HTTP Content-Type header to be used when sending the binary data to the browser. It serves to tell the browser that the transferred binary data represent a photo, a certificate, or whatever.

The request parameters are encoded based on UTF-8.

A download URI can, for example, be assigned to the src attribute of an image tag, the href attribute of a link tag, or to the JavaScript property window.location.href.

6.3.1.1. Properties

The following table lists the download renderer properties:

Name	Value
servletPath	The servlet path of the binary reader servlet as configured in web.xml.
contentType	The value for the HTTP Content-Type header used when downloading a binary property value to the client.
prefix	The extra path information prefix used when downloading a binary property value to the client.
extension	The extra path information extension used when downloading a binary property value to the client.
max	The maximum number of binary property values to process. Specify 0 for all values.
downloadEnabled	Whether to generate a value with download URIs (true), or to set the value to an empty array (false). The default is true. If the array is empty, the certificate renderer does not display download icons along with certificates.

6.3.1.2. Samples

If no binary attribute value is available, the renderer class sets the value property to an empty array:

```
[]
```

If one photo is available, the renderer class sets the value property to an array of length 1:

```
["binaryReader/photo_0.gif?dn=cn%3DAbele+Marc%2Cou%3DFinances%2Co%3DMy-Company%2Ccn%3DUsers%2Ccn%3DMy-Company&attr=photo&index=0&contentType=image%2Fgif"]
```

6.3.2. X509CertificateRenderer

The renderer generates a JavaScript representation of a user's (or any other object's) certificates. It sets three properties that can be evaluated by JavaScript code in a corresponding HTML renderer snippet:

- **labels** - A JavaScript array with the labels of the fields returned per certificate
- **certificates** - A JavaScript array whose items are again arrays. Each item array lists the component values for a single certificate.
- **value** - A JavaScript array with the download URIs for the certificates (see section "DownloadRenderer").

6.3.2.1. Properties

The class supports the following properties in addition to the properties of the DownloadRenderer:

Name	Value
attributes	A comma-separated list of field names whose values are to be returned per certificate. The fields are returned in the same order as listed here. See the table below for available field names. For details on the fields, see the Java documentation of class java.security.cert.X509Certificate.
dateFormat	The requested format for dates. To get localized dates, provide the message key of a date format. See the Java documentation of class java.text.SimpleDateFormat for details on date formats. Default: MM/dd/yyyy z
dnFormat	The requested format for DNs. For details, see the Java documentation class javax.security.auth.x500.X500Principal. Default: RFC1779
messagePrefix	The message prefix for the labels to be returned. The label's message key is the concatenation of the prefix, a dot and the field name (for example, certificate.issuerDN). Default: certificate

Name	Value
timeZone	The requested time zone for dates. Note that your users may work in different time zones. Default: The local time zone of the Web server

6.3.2.2. Certificate Field Names

See the Java documentation of class java.security.cert,X509Certificate for details on the listed Java methods.

Name	Java-script Type	Java Method / Notes
basicConstraints	String	`X509Certificate.getBasicConstraints()`
certificate	String	`X509Certificate.toString()`
encoded	String	`X509Certificate.getEncoded()`
issuerDN	String	`X509Certificate.getIssuerX500Principal().getName(format)` Formatted according to renderer class property dnFormat.
issuerUniqueID	String	`X509Certificate.getIssuerUniqueID()`
keyUsage	String	`X509Certificate.getKeyUsage()`
notAfter	String	`X509Certificate.getNotAfter()` Formatted according to renderer class properties dateFormat and timeZone.
notAfterMs	Number	`X509Certificate.getNotAfter().getTime()` Number of milliseconds since January 1, 1970, 00:00:00 GMT. Can be used to display the date according to the end user's time zone.
notBefore	String	`X509Certificate.getNotBefore()` Formatted according to renderer class properties dateFormat and timeZone.
notBeforeMs	Number	`X509Certificate.getNotBefore().getTime()` Number of milliseconds since January 1, 1970, 00:00:00 GMT. Can be used to display the date according to the end user's time zone.
publicKeyAlgorithm	String	`X509Certificate.getPublicKey().getAlgorithm()`

Name	Java-script Type	Java Method / Notes
publicKeyEncoded	String	`X509Certificate.getPublicKey().getEncoded()`
publicKeyFormat	String	`X509Certificate.getPublicKey().getFormat()`
serialNumber	String	`X509Certificate.getSerialNumber()`
sigAlgName	String	`X509Certificate.getSigAlgName()`
sigAlgOID	String	`X509Certificate.getSigAlgOID()`
sigAlgParams	String	`X509Certificate.getSigAlgParams()`
signature	String	`X509Certificate.getSignature()`
subjectDN	String	`X509Certificate.getSubjectX500Principal().getName(format)` Formatted according to renderer class property dnFormat.
subjectUniqueID	String	`X509Certificate.getSubjectUniqueID()`
TBSCertificate	String	`X509Certificate.getTBSCertificate()`
type	String	`X509Certificate.getType()`
valid	Boolean	`X509Certificate.checkValidity()` true if the certificate is currently valid, false otherwise
version	String	`X509Certificate.getVersion()`

6.4. Configuration in web.xml

This section provides information about configuration in web.xml.

6.4.1. The Binary Reader Servlet

The binary reader servlet processes requests for binary data. It delegates the task to read the data from the LDAP directory to the JSP /WEB-INF/jsp/controller/binary/default.jsp. The path to the JSP must be configured here. There should be no need to customize the JSP.

6.4.1.1. Definition

```
<servlet>
    <servlet-name>BinaryReaderServlet</servlet-name>
    <servlet-class>
        siemens.dirxjsp.servlet.BinaryReaderServlet
    </servlet-class>
    <init-param>
```

```
        <param-name>JspPath</param-name>
        <param-value>
            /WEB-INF/jsp/controller/binary
        </param-value>
    </init-param>
</servlet>
```

6.4.1.2. Mappings

The mapping defines the servlet path for the binary reader servlet.

```
<servlet-mapping>
    <servlet-name>BinaryReaderServlet</servlet-name>
    <url-pattern>/binaryReader/*</url-pattern>
</servlet-mapping>
```

6.4.2. The Binary Request Filter

The binary request filter is a variant of the request filter used for all other Web Center requests. The only difference is that the filter itself takes care of correct request parameter decoding. The request encoding must be UTF-8 since the DownloadRenderer class encodes request parameters based on this encoding.

6.4.2.1. Definition

```
<filter>
    <filter-name>BinaryRequestFilter</filter-name>
    <display-name>Binary Request Filter</display-name>
    <description>Binary Request Filter</description>
    <filter-class>
    siemens.dirxjsp.core.application.RequestFilter
</filter-class>
    <init-param>
        <param-name>CleanUpEnabled</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>RequestSyncEnabled</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
```

```
        <param-name>RequestEncoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
    <init-param>
        <param-name>IgnoreLocale</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>DecodeRequestParameters</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
```

6.4.2.2. Mappings

The filter must be mapped to all requests served by the binary reader servlet.

```
<filter-mapping>
    <filter-name>BinaryRequestFilter</filter-name>
    <url-pattern>/binaryReader/*</url-pattern>
</filter-mapping>
```

6.4.3. The AddHeaderFilterForDownloads Filter

The filter is used to direct the browser to keep downloaded photos (and other binary attributes) in its cache for a period of time. This action prevents the browser from downloading a photo whenever the user clicks the download icon or hovers over the photo thumbnail.

The default caching period is 10 minutes (600 seconds).

6.4.3.1. Definition

```
<filter>
    <filter-name>AddHeaderFilterForDownloads</filter-name>
    <display-name>
    Add Header Filter for Downloads
</display-name>
    <description>
    Adds headers to HTTP response when downloading files
</description>
    <filter-class>
```

```
    siemens.dirxjsp.core.application.AddHeaderFilter
</filter-class>
    <init-param>
        <param-name>Expires-Header</param-name>
        <param-value>http/1.*:Expires:now+600</param-value>
    </init-param>
</filter>
```

6.4.3.2. Mappings

The filter must be mapped to all requests served by the binary reader servlet.

```
<filter-mapping>
    <filter-name>AddHeaderFilterForDownloads</filter-name>
    <url-pattern>/binaryReader/*</url-pattern>
</filter-mapping>
```

6.5. Approving Photos and Certificates

Web Center supports the approval of photos and certificates in request workflows and tickets. An approver can approve or reject an assignment, but he may not assign a different photo or certificate.

For example, if modification approval is activated for photos and a user assigns a photo to another one, a request workflow is started to let some approvers approve the assignment. Or if modification approval is activated for certificates and a user assigns one or more certificates to another one with a due date, a ticket is created and a workflow to approve the assignment is started.

You cannot assign a photo or certificate to an object in an **enterAttributes** activity of a request workflow. But if you manage somehow to assign a photo or certificate in an automatic activity you can let some approvers approve the assignment in a later approval activity of the same workflow. Just add the attribute name (like **photo** or **userCertificate**) to the attributes list on the **parameters** page of the activity.

6.5.1. Modification Approval

The renderers for binary attributes on modification approval pages are defined in file **WEB-INF/config/workflows/modificationApproval/forms-config.xml**:

```
<form-bean name="approveUserModificationActivityForm">
  …
  <form-property name="userCertificate"
        type="java.lang.Object[]"
```

```
        readonly="true"
        fieldRenderer="certificates"
        rendererProperties="downloadEnabled:false"
      y="+1"/>
  <form-property name="jpegPhoto"
        type="byte[]"
        fieldRenderer="photo"
        readonly="true"
        y="+1"/>
  <form-property name="photo"
        type="java.lang.Object[]"
        fieldRenderer="photos"
        readonly="true"
        y="+1"
        spanX="4"
        hideOldIfEmpty="true"/>
```



- The form property attribute **readonly** must be set to true.
- The property type depends on whether the property is single- or multi-valued. Use type byte[] for single-valued properties, type java.lang.Object[] for multi-valued properties.
- You can have the certificate renderer hide download icons by setting its property **downloadEnabled** to false.
- New and old values of the last attribute **photo** are displayed below each other due to spanX="4". If the attribute does not have any old value, the label “Old value” is not displayed due to hideOldIfEmpty="true".

6.5.2. Creation Approval

The renderers for binary attributes on all other approval pages are defined in file **WEB-INF/config/defaultRenderer.properties**:

```
jpegphoto       = photo
#jpegphoto      = photos
photo           = photos
#photo          = documents
usercertificate = certificates
```



- The list maps property names to renderer identifiers.
- Use lower-case property names.

- Usually, all approval pages use the same renderer for a property. The only choice is to assign renderers per object class, for example

```
photo          = photos
photo@dxruser = documents
```

7. DN Representation

This chapter describes how to configure the representation of distinguished names (DNs) within Web Center.

7.1. Overview

Web Center distinguishes between a normal DN representation and a short form. The short form is used to save space when rendering role parameter values of type DN or hierarchical DN in tables and read-only fields.

The representations are controlled by a set of configuration parameters defining the name parts to be included, the delimiters between name parts, whether to include name part types (like cn and ou) etc.

The name parts to be included may be overridden per renderer or per form property.

The configurable representations include:

```
cn=Taspatch Nik,ou=Global IT,o=My-Company,cn=Users,cn=My-Company
Taspatch Nik, Global IT, My-Company, Users, My-Company
Taspatch Nik, Global IT, My-Company
/cn=Users/o=My-Company/ou=Global IT/cn=Taspatch Nik
Users --> My-Company --> Global IT --> Taspatch Nik
```

7.1.1. Terms

For the DN

```
cn=Taspatch Nik,ou=Global IT,o=My-Company,cn=Users,cn=My-Company
```

we define:

- The bottommost or first node is cn=Taspatch Nik.
- The topmost or last node is cn=My-Company.

The terms name part and node are used synonymously.

7.2. Configuration

The configuration parameters are defined in two files, one of which is evaluated in the browser while the other one is utilized on the server.

7.2.1. Client-Side Configuration

The configuration parameters in the Javascript file **resources/build/config/config.js** are evaluated by Javascript code running in the browser. The parameters control the representation of DNs in the user interface.

7.2.1.1. Configuration Parameters

The parameters are stored in section **dn**:

- **delim –** String; the delimiter between adjacent nodes; default: ", ".
- **excludeTopNodes –** Number; the number of top nodes to be excluded from the representation; default: 1.
- **oldStyle –** Boolean; whether to use the traditional DN representation; if true all other configuration parameters are ignored; default: false.
- **prefix –** String; a prefix for the representation; default: the empty string.
- **reversed –** Boolean; whether to list the nodes in reversed order (from top to bottom); default: false.
- **types –** Boolean; whether to display name part types; default: false.

Additional parameters controlling the short DN representation are stored in subsection **shortForm**.

- **ellipsis –** String; an appendix to the representation indicating an abbreviated form; default: " ...".
- **includeNodes –** A number array; the nodes to be included in the representation (for details see below); default: [0,1].

7.2.1.2. Parameter includeNodes

The value for parameter **includeNodes** is a number array of length 1 or 2:

- **[bottomIndex] –** The representation includes all name parts from the bottom index to the topmost name part.
- **[bottomIndex,topIndex] –** The representation includes all name parts starting from the bottom index and ending just before the top index.

The index of the bottommost node is 0. Negative indexes indicate offsets from the top; -1 denotes the topmost name part.

Samples:

- [0] – All name parts.
- [1] – All name parts besides the first one, which gives the parent DN.
- [0,1] – Just the first node.
- [0,-1] – All nodes besides the topmost one.

7.2.1.3. Parameter Evaluation Order

1. Apply parameter **excludeTopNodes**.
2. Apply parameter **includeNodes**.
3. Apply parameter **reverse**.
4. Apply parameters **delim** and **prefix**.

The other parameters are order independent.

7.2.1.4. Samples

Sample DN:

```
cn=Taspatch Nik,ou=Global IT,o=My-Company,cn=Users,cn=My-Company.
```

1. The default configuration:
   - **dn.delim**: “, “
   - **dn.excludeTopNodes**: 1
   - **dn.prefix**: “”
   - **dn.reversed**: false
   - **dn.types**: false
   - **dn.shortForm.ellipsis**: “…”
   - **dn.shortForm.include**: [0,1]

   The DN representation is

   ```
   Taspatch Nik, Global IT, My-Company, Users.
   ```

   The abbreviated DN representation is

   ```
   Taspatch Nik ...
   ```

2. A customized configuration:
   - **dn.delim**: “/“
   - **dn.excludeTopNodes**: 2
   - **dn.prefix**: “/”
   - **dn.types**: true
   - **dn.reversed**: true

- **dn.shortForm.ellipsis**: “”
- **dn.shortForm.includeNodes**: [0,-1]

The DN representation is

```
    /o=My-Company/ou=Global IT/cn=Taspatch Nik.
```

The abbreviated DN representation is

```
    /ou=Global IT/cn=Taspatch Nik.
```

3. The old-style configuration:
   - **dn.oldStyle**: true

   DN and abbreviated DN representation are

```
    cn=Taspatch Nik,ou=Global IT,o=My-Company,cn=Users,cn=My-
Company.
```

7.2.2. Server-Side Configuration

The configuration parameters in the file **WEB-INF/config/webCenter.properties** are evaluated by Web Center code running on the server. The parameters control the representation of DNs in export files.

7.2.2.1. Configuration Parameters

The supported parameters are a subset of the client-side ones, with identical meanings and default values:

- dn.delim
- dn.excludeTopNodes
- dn.oldStyle
- dn.prefix
- dn.reversed
- dn.types

7.2.2.2. Sample

The default configuration is:

- **dn.delim** = ,\u0020

- **dn.excludeTopNodes** = 1
- **dn.oldStyle** = false
- **dn.prefix** =
- **dn.reversed** = false
- **dn.types** = false

7.2.3. Renderer Configuration

Some DN renderers allow for overriding the globally defined value for the **includeNodes** parameter.

7.2.3.1. Affected Renderers

The **includeNodes** option is evaluated by the renderer snippets

- dn.htm
- dnList.js
- roDN.js

The option is not supported by the renderer snippet

- dn.js

The renderers based on the listed snippets are

- **dn –** The renderer is used for single-valued DN form fields. It is based on the snippets dn.htm and dn.js.
- **dnCombobox –** The renderer is used for role parameter values of type hierarchical DN. It is based on the snippets dn.htm and roDN.js.
- **dnList –** The renderer is used for multi-valued DN properties like role owner. It is based on the snippets roStringList.htm and dnList.js.
- **roDN –** The renderer is used for read-only form fields like parent folder of a role. It is based on the snippets dn.htm and roDN.js.

7.2.3.2. Renderer Parameter

- **includeNodes –** A number array; the nodes to be included in the representation (for details see configuration file **config.js**).

7.2.3.3. Sample

```
<renderer id="roDN"
          defURL="/WEB-INF/snippets/dn/dn.htm"
          jsURL="/WEB-INF/snippets/dn/roDN.js">
<renderer-property name="includeNodes" value="[0]"/>
```

```
<renderer-property name="link"          value="false"/>
</renderer>
```

7.2.3.4. Form Property Configuration

Form properties rendered by one of the renderers listed above allow for overriding the **includeNodes** parameter value defined for the renderer.

The **includeNodes** option can be passed from the form property to the renderer via the form property attribute **rendererProperties** that serves to set renderer properties in a generic way.

7.2.3.5. Sample

```
<form-property name="$parentfolder" type="java.lang.String"
        fieldRenderer="roDN"
        rendererProperties="includeNodes:[0,-1]"
        label="ldap.pseudoattribute.path" .../>
```

The DN representation of the parent folder includes all nodes but the topmost one.

7.3. Individual DN Representations

In some situations it's preferable to render a distinguished name in a way different from the default representation.

7.3.1. Single-valued DN attributes

For single-valued DN attributes, Web Center provides the renderer **path**. The renderer lets you define

- **delim –** The name part delimiter; default: /.
- **includeNodes –** The name parts to include; default: [0].
- **reversed –** Whether to display the name parts in reverse order; default: true.
- **link –** When used to render a DN in a table column: Whether clicking a DN opens the entry displayed in the table row. Default: false.

The default values are assigned to the properties of the renderer in file **renderers-config.xml**.

The renderer can be used to display single-valued DN attributes in form fields or table columns. Editing attributes is not supported.

7.3.1.1. Samples

7.3.1.1.1. Form Property Definition

```
<form-property name="dxrWorkflowLink"
    type="java.lang.String"
    readonly="true"
    fieldRenderer="path"
    rendererProperties="delim:-->" .../>
```

Sample output

```
wfRoot-->Definitions-->Default-->Assignments-->4-Eye Approval
```

7.3.1.1.2. Data Property Definition

```
<data-property name="dxrWorkflowLink"
    type="java.lang.String"
    cellRenderer="path"
    rendererProperties="link:true;reversed:false" .../>
```

Sample output:

```
4-Eye Approval/Assignments/Default/Definitions/wfRoot
```

7.3.2. Multi-valued DN attributes

For multi-valued DN attributes, Web Center provides the renderer **pathList**. The renderer let's you define

- **itemRenderer –** The id of the underlying single-valued DN renderer; default: **path**.

The renderer can be used to display multi-valued DN attributes in form fields. Displaying attributes in table columns and editing attributes is not supported.

Note that you cannot overwrite the properties of the underlying renderer (**delim**, **includeNodes**, **reversed**) via properties of the **pathList** renderer or the **rendererProperties** attribute of the form properties. All you can do is assign a customized copy of the **path** renderer as item renderer.

7.3.2.1. Samples

7.3.2.1.1. Form Property Definition

```
<form-property name="owner"
    type="java.lang.String[]"
    readonly="true"
    fieldRenderer="pathList" .../>
```

Sample output:

```
Users/My-Company/Global IT/Taspatch Nik
Users/My-Company/Human Resources/Telfer Laura
```

8. Attribute Modification Approval

This chapter describes how to configure attribute modification approval within Web Center.

8.1. Overview

Web Center supports modification approval for a wide range of objects and attributes. The content and layout of the approval pages is controlled by a single forms configuration file.

The configuration allows for:

- Displaying the attributes to be approved in a defined order.
- Arranging the attributes in sections with titles.
- Defining which attributes the approver is allowed to modify.
- Defining display properties like field width and renderer per attribute.
- Displaying localized attribute names.
- Displaying old and new attribute values side by side on the same row, or one below the other on subsequent rows.
- Displaying non-modified attributes that are somehow related to one of the modified attributes.
- Defining which modified attributes are hidden from the approver.
- Assigning checks to attributes.

8.2. Form Configuration

The configuration is split into a base form containing the common parts of all approval forms, and of forms specific to object types and/or approval workflows. The specific forms describe the actual attributes to be approved, and are included into the base form at runtime as appropriate.

The forms configuration file is

> **WEB-INF/config/workflows/modificationApproval/forms-config.xml**.

After changes to the file, restart Tomcat to put the changes into effect.

8.2.1. Base Form Configuration

The base approval form configuration is the **form-bean** element with name **approveModificationActivityForm**. It is a variation of the usual form configuration. It defines the common fields to be displayed on the top and the bottom of all approval pages, like workflow subject, initiator, expiration time, and reason, as well as the buttons to accept

or reject modifications. In addition to that, the base form contains elements that define which forms to include at runtime for which object types and approval workflows

8.2.1.1. Top and Bottom

Top and bottom are configured as usual. The only thing to pay regard to is that the top fields get sufficiently low **y** values and the bottom fields get sufficiently high ones, so that the fields in the forms to be included at runtime get **y** values that lie between the top and the bottom ones.

8.2.1.2. Includes

Form-include elements are instructions to include a form at runtime into the base form. The element supports the attributes **name**, **objectTypes** and **workflowPaths**.

The **name** is the name of the **form-bean** element that is to be included at runtime at the element's position.

The **objectTypes** are a comma-separated list of case insensitive object types this include applies to. An **include** element with an empty **objectTypes** attribute applies to any object.

The **workflowPaths** are a comma-separated list of Java regular expressions. An **include** element applies only to those approval workflows whose path case insensitively matches at least one of the expressions. An **include** element with an empty **workflowPaths** attribute applies to any workflow.

Samples:

```
<form-include
     name="approveUserModificationActivityForm"
     objectTypes="dxrUser"
     workflowPaths=""/>
```

The include element matches any approval workflow for **dxrUser** subjects.

```
<form-include
     name="approvePermissionModificationActivityForm"
     objectTypes="dxrPermission"
     workflowPaths=".*/Modify Permission$"/>
```

The **include** element matches any workflow for **dxrPermission** subjects whose path ends with **/Modify Permission**.



The path of a workflow is its distinguished name in reversed order, with delimiter "/", without types and without the workflow root DN:

Workflow DN:

```
cn=Modify User,cn=Users,cn=Default,
       cn=Definitions,cn=wfRoot,cn=My-Company
```

Workflow path:

```
/Definitions/Default/Users/Modify User
```

8.2.2. Included Forms

The included forms are regular **form-bean** elements whose child elements define layout and properties of the attributes to be approved, and sections that serve to structure the approval page with titles.

8.2.2.1. Section

A section is a **form-property-group** element with a unique name. The message key of its title is by default **group.label.**<case insensitive group name>, but may be set explicitly in attribute **label**.

Each section has a level, a non-negative number. The level is specified in attribute **level**; the default level is **0**. A section is regarded to be a logical subsection of the last preceding section with a lower level.

A section title is displayed only if at least one of the attributes in the section or a logical subsection thereof must be approved. A section extends to the next **form-property-group** element with the same or a lower level, or the end of the form.



Do not define any **form-property** child elements for sections.

8.2.2.2. Attributes

Attributes are configured as usual via **form-property** elements supporting almost any attribute in the familiar way. But there are some differences.

The message key of the attribute label is by default **ldap.attribute.**<case insensitive form-property name>, but may be set explicitly in attribute **label**.

For each attribute to be approved, the **spanX** attribute defines whether old and new values are displayed side by side on the same row (**spanX** <= 2), or old below new values on two subsequent rows (**spanX** > 2). In the latter case, old and new values span three columns each.

In case a property does not have any old value and its **spanX** attribute value is 4, the label for the old property values is displayed anyway, wasting space for a separate row. You can suppress the label display by setting attribute hideOldIfEmpty to true. It is sometimes

convenient to display related attributes along with an attribute that must be approved even if the related attributes were not modified or are not subject to approval. For example, if someone changed a user's disable start date, the approver might prefer to see the disable end date as well. This can be achieved with attribute sets.

A set is identified by an arbitrary unique id and comprises several attributes of the same section. An attribute belongs to all sets whose ids are listed in its **form-property** attribute **sets**. If a modification of any of the attributes in a set must be approved, the approval page shows all attributes in that set. A set attribute that was not modified or is not subject to approval is editable for the approver only if the set id in its **sets** list is prefixed with **w:**

Sets are esp. useful for defining field checks that affect more than one attribute.

Sample:

```
<form-property name="dxrStartDate" ... sets="d"/>
<form-property name="dxrEndDate" ... sets="d"/>
<form-property name="dxrDisableStartDate" ... sets="w:d"/>
<form-property name="dxrDisableEndDate" ... sets "w:d"
     fieldRenderer="checkedDate"
     rendererProperties="onChange:dates.checkOrder;
     form:approveModificationActivityForm;

ids:dxrStartDate,dxrDisableStartDate,dxrDisableEndDate,dxrEndDate"
/>
```

The attribute set with id "d" consists of four attributes. The approver will see either all four attributes or none at all. If displayed, dxrDisableStartDate and dxrDisableEndDate are always editable for the approver, dxrStartDate and dxrEndDate only if originally modified and subject to approval.

8.2.3. Generic Attribute Display

Web Center will attempt to display attributes subject to approval but not listed in the forms configuration on the approval page in a simple generic way below the other attributes. The generic approach, however, works only for a limited subset of all attributes, and lacks many of the features of the configuration approach. Therefore, we strongly recommend adding each affected attribute to the approval form configuration.

8.3. Restrictions

Attribute modification approval is, amongst others, not possible for

- Naming attributes.
- The assigned state of group members.

You must not activate approval for these attributes.

8.4. Notes

Some modifications affect several attributes of an object. In that case, you must either activate approval for all affected attributes or for none at all. If activated, list each affected attribute in the approval form configuration, and set the **visible** flags of the implicitly changed attributes to **false** in order not to confuse the approver. Examples are:

- Role parameter changes affect attributes dxrRPMatchRule and dxrRoleParamLink.
- Changes to the group flag “Manage only in target system” sometimes causes the dxrError attribute to be cleared.

9. Localizing Properties with a Finite Number of Values

This chapter shows, by means of a sample, how to localize the values of a property with a finite number of values.

As our sample property we choose request workflow activity state. The state of an activity is one of

- FAILED.ABORTED
- FAILED.EXPIRED
- RUNNING
- SUCCEEDED

First, define a message prefix for the property, let's say

- requestwf.activitystate

The prefix should be unique in file **text.properties**.

Then, for each supported language, assign a localized text to each property value in file

WEB-INF/classes/resources/languages/<language>/text.properties.

The key for each value is

<message prefix>.<lower case property value>.

Let's for example define the following english texts for the workflow activity states:

- requestwf.activitystate.failed.aborted = Cancelled
- requestwf.activitystate.failed.expired = Expired
- requestwf.activitystate.running = Running
- requestwf.activitystate.succeeded = Succeeded

Finally, assign the prefix to the **messagePrefix** attribute of each **form-property** and **data-property** element for the property, e.g.

```
<form-property name="actstate" readonly="true"
    type="java.lang.String" label="requestwf.activitylist.state"
    width="180" messagePrefix="requestwf.activitystate"/>
```

and

```
<data-property name="actstate"
```

```
    type="java.lang.String" label="requestwf.activitylist.state"
    sortIndexes="3,0"  width="14%"
messagePrefix="requestwf.activitystate"/>
```

10. Customizing the Overall Page Layout

10.1. Files

File with default settings:

WEB-INF/config/webCenter.properties

Customization file:

WEB-INF/config/webCenterCustom.properties

10.2. Layout Properties

This section lists the layout configuration parameters. Note that some of the possible parameter value combinations will break the table-based page layout and still require manual adjustments in some view JSPs.

10.2.1. Header

This section contains some properties affecting the layout of the header. The header is split into a top section above the horizontal menu bar and a small bottom section below the menu bar.

The top section includes the options panel with language and font size chooser, and the links panel with links to logout or to cancel a registration.

If a panel or a section is hidden, all of its subcomponents are hidden as well.

headerTop	true – Show top section; default. false – Hide top section.
headerKeyVisual	true – Show keyvisual in top section; default. false – Hide keyvisual.
headerCompanyLogo	true – Show logo in top section; default. false – Hide logo.
headerCompanyName	true – Show company name in top section; default in the case logo is hidden. false – Hide name; default in case the logo is visible.
headerProductName	true – Show product name in top section; default. false – Hide product name.

headerWelcome	true – Show welcome text in top section; default. false – Hide welcome text.
headerLinks	true – Show links panel in top section; default. false – Hide links.
headerLogoutLink	true – Show logout link in links panel. false – Hide logout link; default.
headerOptions	true – Show options panel in top section; default. false – Hide options area.
headerOptionsLanguage	true – Show language chooser in options panel; default. false – Hide language chooser.
headerOptionsFontSize	true – Show font size chooser in options area; default. false – Hide font size chooser.
headerBottom	true – Show bottom section; default. false – Hide bottom section.

10.2.2. Footer

This section contains some properties affecting the footer layout. Product version and suite are not shown by default for security reasons.

footer	true – Show footer; default. false – Hide footer.
footerRuler	true – Show ruler; default. false – Hide rule.
footerLeft	true – Show product version. false – Hide product version; default.
footerCenter	true – Show copyright; default. false – Hide footer.

footerRight	true – Show product suite. false – Hide product suite; default.

10.2.3. Menu Position

The menu may be displayed either horizontally, or vertically on the left of each page.

menuVertical	true – Show vertical menu on the left. false – Show horizontal menu; default.

10.2.4. Font Size

The default font size can be set via property “defaultStyle”. It is especially important if you don’t let your users choose a font size via the font size chooser in the options area.

availableStyles	The list of supported font size names. Requests to change the font size to a different value are ignored (for security reasons). Default: “large,medium,small”
defaultStyle	small – Use CSS stylesheet for small font size. medium – Use CSS stylesheet for medium font size. large – Use CSS stylesheet for large font size.

10.2.5. Resource Files

This section defines some path names of the application’s resources. Specify path names relative from the application’s root folder.

Since resource files are directly downloaded by the browser, you cannot put them somewhere below the **WEB-INF** folder.

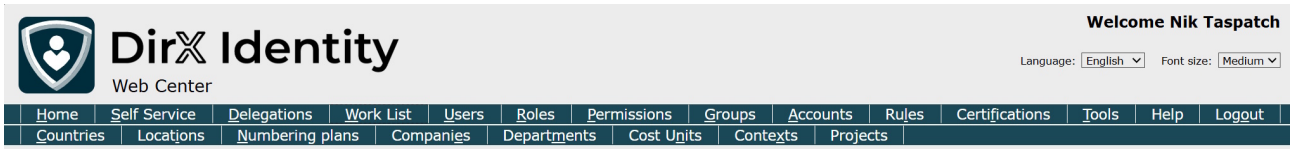
resourceFavicon	The name of the application’s favorite icon. The icon is displayed in the browser’s address bar and on browser tabs, as well as for shortcuts to Web Center pages in the file system.
resourceLogo	The application logo in the header top section.

The size of the default application logo is 302 by 52 pixels. If your logo is of a different size it will be re-sized for display and might look distorted. You can adjust the display size to some extent via CSS styles “headerCompanyLogo” and “headerCompanyLogoImage”.

10.3. Customizing the Layout - Step by Step

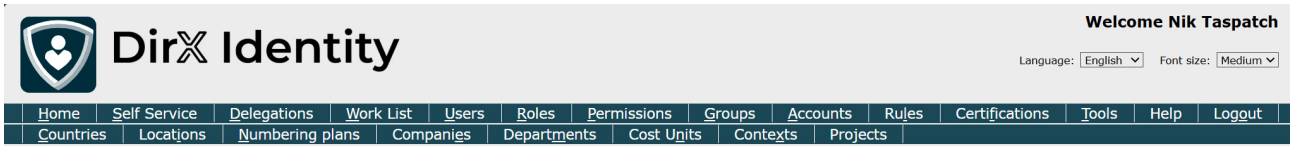
10.3.1. Header

10.3.1.1. Standard Layout



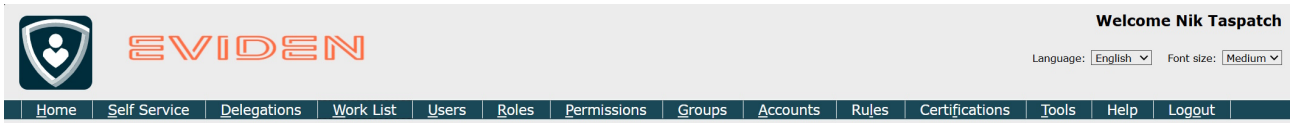
10.3.1.2. Hide Product Name

headerProductName	false



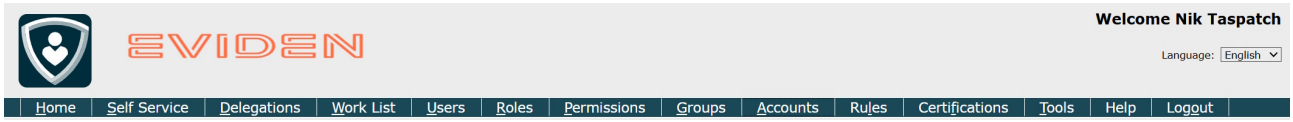
10.3.1.3. Replace Company Name with Company Logo

headerBrandLogo	false
headerCompanyLogo	true
headerCompanyName	false
resourceLogo	.. path to company logo ..



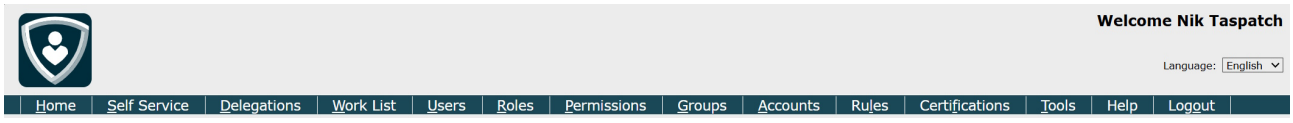
10.3.1.4. Hide Font Size Option

headerOptionsFontSize	false



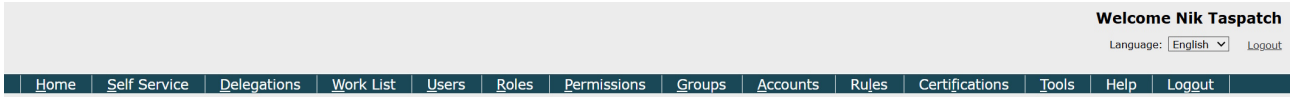
10.3.1.5. Hide Company Name and Logo

headerCompanyLogo	false
headerCompanyName	false



10.3.1.6. Hide Keyvisual and Show Logout Link

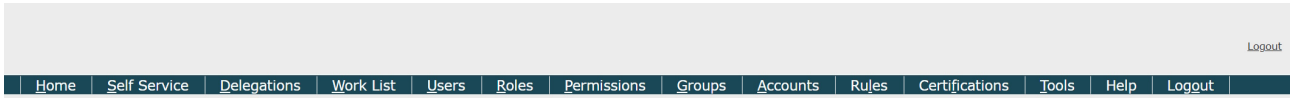
headerKeyVisual	false
headerLogoutLink	true



The logout menu can be disabled in the menu configuration file. See the chapter on menus for details.

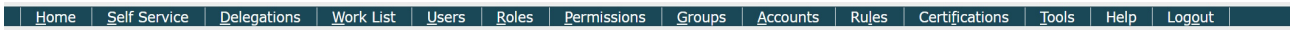
10.3.1.7. Hide Options and Welcome Message

headerOptions	false
headerWelcome	false



10.3.1.8. Hide Top and Bottom Part of Header

headerTop	false
headerBottom	false



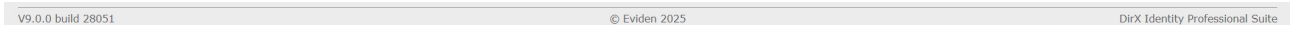
10.3.2. Footer

10.3.2.1. Standard Layout



10.3.2.2. Show Version and Suite

footerLeft	true
footerRight	true



10.3.2.3. Hide Ruler and Copyright

footerRuler	false
footerCenter	false

10.3.2.4. Hide Entire Footer

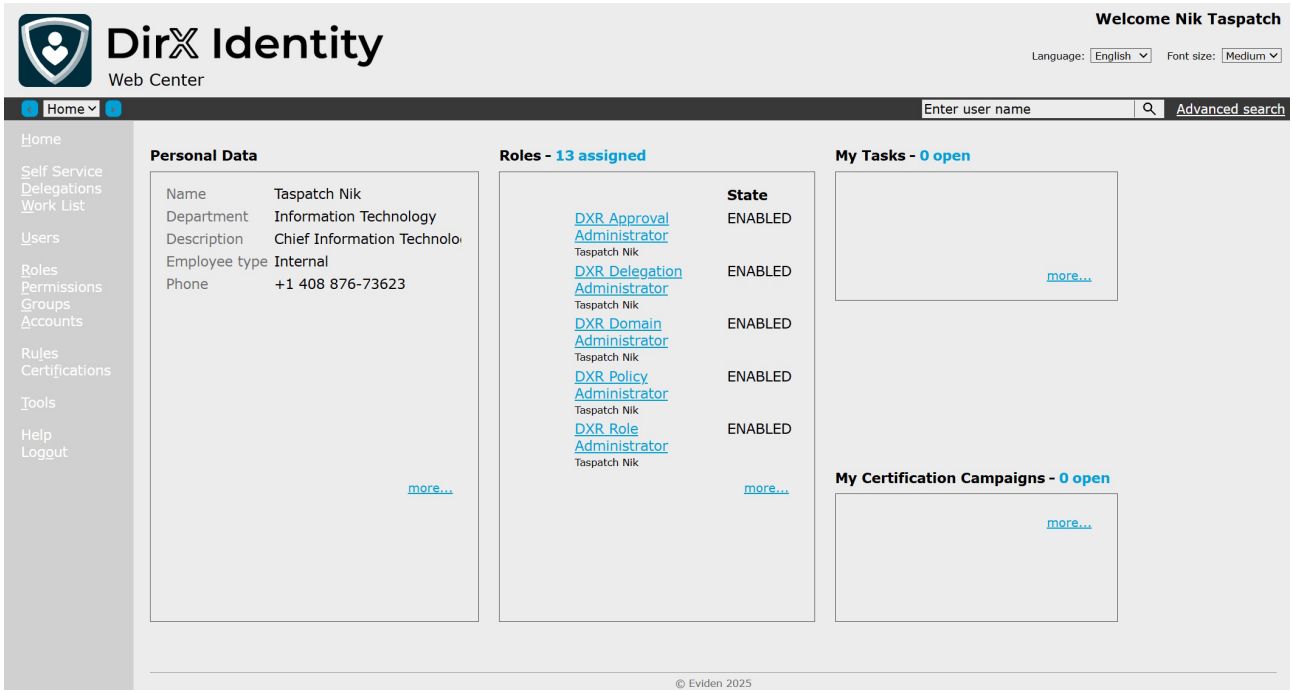
footer	false

10.3.3. Menu

10.3.3.1. Display Menu Vertically

Web Center may display the menu bar on the left side of the page instead of in the header section:

menuVertical	true



Adapt the menu bar width via appropriate CSS style definitions in **styles.css**.

11. Web Center Menus

This chapter provides information about Web Center menus.

11.1. Menu Overview

This section provides an overview on Web Center menus.

11.1.1. Menu Access Policies

Menu access policies control access of a user to menus and menu items in the navigation bar, and to context menus and their items.

Menu access policies are no substitutes for operational access policies.They do not define which operations a user is allowed to perform, and on which objects.For example, a user may be granted access to the “modify role” menu item while not being allowed to modify any role at all.On the other hand, a user who is denied access to the “modify role” menu item might still be able to modify roles via other means like clicking a respective button or image, by being forwarded to the “modify role” page as result of another action, or by typing the “modify role” URL in his browser’s address bar.

Menu access policies are derived on the basis of object types (like “role”) but for performance reasons not on the basis of specific objects.That means, for example, if a user is granted access to the “delete role” menu item, he is able to select that item on any role, no matter whether or not he is actually allowed to delete it.

Note: Though menu access policies were designed to control access to menu items, they can sometimes be leveraged for different purposes as well. Web Center, for example, controls visibility of open buttons in tables via menu access policies. The <ctrl:checkAccessPolicy> tag allows for checking whether the logged in user is granted the right to a given menu item.

11.1.2. Navigation Menu

This section provides information about the navigation menu.

11.1.2.1. Menu Handling

The menus in the navigation bar can be operated by mouse and keyboard.

A menu is opened by clicking on its title, or by entering its access key. A menu access key is entered by pressing by Shift + Alt + <key>. Depending on the browser Alt + <key> alternatively may also work, but can be overridden by the browser’s own access keys. The keys are case insensitive.

Once open, you can browse thru the entire navigation bar by mouse movements or cursor keys (up, down, left, right).

Moving the mouse onto an item and clicking the main mouse button or pressing the

RETURN key selects it. An item gets also selected by pressing its (case insensitive) access key.

A menu is closed by clicking somewhere outside the menu, or by pressing the ESCAPE key.

11.1.2.2. Menu Definition

A menu is defined in the application's /WEB-INF/config/identity/menu-defs.xml file as a menu bar definition, for example

```
<definition name=".defaultMenubar">
    <put name="items" value="..."/>
</definition>
```

For details, see the chapter Menu Configuration in the DirX Identity Web Center Reference.

Menus may be also defined in additional menu-defs.xml files which must then be added to the application's context descriptor file.

11.1.2.3. Menu Labels

A menu configuration contains message keys for menu titles and menu item labels. The message text may contain an ampersand indicating that the next letter should be used as access key for that menu or item.

Access keys are case insensitive. Do not use the same access key for more than one menu, or for more than one item in a menu.

A message text may contain a placeholder to be replaced at runtime by the value of the session variable defined in the menu item's attribute **session variable**. The supported placeholder formats are

- **{[n]s}** – String; the variable value is converted to a string which is substituted for the placeholder. An optional number defines the maximum number of characters to substitute.
- **{[n]dn}** – DN; the variable value is expected to be a distinguished name. The placeholder is substituted by the value of the first RDN. An optional number defines the maximum number of characters to substitute.

Samples:

The following lines define the title (with access key "r") and two items (with access leys "l" and "d", respectively) for a menu:

```
roles.title         = &Roles
roles.lastSelection = &Last selection list: {30s}
roles.summary       = &Display summary: {30dn}
```

With variable values

- "(cn=sa*)" and
- "cn=Sales Tasks,cn=Corporate Roles,cn=RoleCatalogue,cn=My-Company""",

the item labels resolve to

- "&Last selection list:(cn=sa*)" and
- "&Display summary: Sales Tasks",

respectively.

11.1.2.4. Navigation Bar Location

Web Center supports horizontal menus spanning one or two rows, and vertical menus. The layout JSPs contain definitions for both, horizontal and vertical menus, and switch between them based on a configuration parameter in webCenter.properties.

The JSP "/WEB-INF/jsp/controller/view/tiles/header/header.jsp" defines two tables each containing a single row with id "menubar" and "menubar2", respectively, the designated location for the horizontal menu:

```
<table cellpadding="0" cellspacing="0">
    <tr id="menubar" role="menubar"/>
</table>
…
<table cellpadding="0" cellspacing="0">
    <tr id="menubar2" role="menubar"/>
</table>
```



The id of the second row must match the id of the first one, followed by number 2.

The JSP "/WEB-INF/jsp/controller/view/layoutPage.jsp" defines a table containing a tbody element with id "menubar", the designated location for the vertical menu:

```
<table cellpadding="0" cellspacing="0">
    <tbody id="menubar" role="menubar"></tbody>
</table>
```

11.1.2.5. Assigning a Menu to a Navigation Bar

For the horizontal menu, the header layout JSP "/WEB-INF/jsp/controller/view/tiles/header/header.jsp" invokes the <view:menu> tag assigning the menu bar with name ".defaultMenubar" to the table row with id "menubar":

```
<view:menu menubarId="menubar" definition=".defaultMenubar"/>.
```

For the vertical menu, the JSP “/WEB-INF/jsp/controller/view/layoutPage.jsp” invokes the tag with attribute vertical set to “true”:

```
<view:menu menubarId="menubar" definition=".defaultMenubar"
vertical=”true”/>
```

The tag is also invoked at other times in order to update the menu during asynchronous requests that update only the content area of a Web Center page. Such requests may have an impact on the menu for example by changing entry selection.

11.1.2.6. Generated Javascript Code

The tag generates Javascript code that is inserted into the response sent to the browser, like:

```
<script ...>menu.init("menubar","/webCenter/",”menu”,[...])</script>
```

The first argument is the navigation bar’s table row id. It tells the browser where to place the menu bar.

The second argument is Web Center’s context path, used when sending a request on selection of an item.

The third argument defines a prefix for the menu and menu item labels which is used by accessibility tools to indicate that a label belongs to a menu.

The fourth argument is an array representation of the menu bar. The labels and confirmation messages are localized, message arguments substituted by their actual values.

The optional fifth argument, a boolean, indicates whether to display a vertical (true) or horizontal menu (false); its default value is false.

11.1.2.7. Menu Display

The browser executes the menu.init function, creates the corresponding DOM elements, resolves ampersands in labels to access keys, and handles events like opening a menu or selecting an item. The menu layout is affected by several CSS classes defined in /resources/build/styles/<fontSize>/styles.css.

11.1.2.8. Requests on Menu Selection

On selection of a menu item, the browser sends a request for the action configured for that item in menu-defs.xml. The request includes several HTTP parameters:

- The parameter “startAction” with value “true” is evaluated by the JSP checkSession.jsp and initiates a session clean-up.
- The parameter “WT” is used for CSRF prevention.
- The parameter “ajax” specifies whether the request should update the entire HTML page or just the content area.
- The parameter “actionConfirmed” specifies whether the end user confirmed the request (for example a request to delete some entries).

The item definition

```
modify:/editUserData.do:selectedUser:mod
```

in menu-defs.xml, for example, corresponds to the request URI

```
http://localhost:8080/webCenter/editUserData.do
```

11.1.3. Context Menus

Context menus apply to tables.

A context menu is comprised of three sections. The methods in the first section apply to the visually highlighted entry (see below), the methods in the second section are applied to all selected table entries (for tables with selection checkboxes only), and the methods in the third section apply to the entire table. All sections are optional.

Context menus can be assigned to most tables, including search result lists and tables displaying attributes of an entry, like a user’s roles or a group’s members. There are, however, some exceptions where context menus conflict with other table features, e.g. during delegation. And attribute tables do not support methods that apply to the selected entries or the entire table.

11.1.3.1. Context Menu Handling

You can display and select context menus with the mouse only.

Placing the cursor on a table entry and clicking the context menu mouse button opens a context menu. The table entry is visually highlighted as long as the context menu is open. Note that the highlighted entry gets not automatically selected, that is the items in the second section apply to it only if its selection checkbox is ticked.

Once open, you can browse through the context menu by mouse movements. Moving the mouse onto an item and clicking the main mouse button select it. A context menu is closed by clicking somewhere outside the menu or by hitting the escape key.

11.1.3.2. Context Menu Definition

Context menus are defined in the application's /WEB-INF/config/identity/menu-defs.xml file, for example

```
<definition name=".contextMenuRoles">
    <put name="accessPolicyKey" value="RoleMgt"/>
    <put name="msgPrefix"       value="ctx.objects"/>
    <put name="items"           value="summary;modify;sep;

assignPrivileges;assignUsers;unassignUsers;sep;
                                    runReport;

ms;delete:confirm.roles,dueDate.delete;
                                    list;export"/>
</definition>
```

For details, see the chapter Menu Configuration in the DirX Identity Web Center Reference.

Note that simply switching an item from one section to another will usually not work since the underlying Struts action must support the respective entry set (highlighted entry, selected entries or entire table).

Context menus may be also defined in additional menu-defs.xml files which must then be added to the application's context descriptor file.

11.1.3.3. Context Menu Assignments

A context menu is assigned to a table in the attribute **contextMenu** of the table's <form-property> element within a forms-config.xml file. The assignment includes the context menu name, and flags to enable the second and third menu section (if applicable). By default, only the first section is enabled. The same context menu can be assigned to different tables, and with different flags.

The following example assigns the first section of the context menu with name ".contextMenuRoles":

```
<form-property name="assignedRoles"
    type="com.siemens.webMgr.model.DirectoryEntryBean[]" ...
    contextMenu=".contextMenuRoles">
```

The following example assigns all sections of the context menu with name ".contextMenuRoles":

```
<form-property name="roles"
    type="com.siemens.webMgr.model.DirectoryEntryBean[]" ...
    contextMenu=".contextMenuRoles:ms,list">
```

11.1.3.4. Generated Javascript Code

The Javascript representation of a context menu is a Javascript array generated as part of the code for the table and inserted into the response sent to the browser.

The HTML code snippet “WEB-INF/snippets/simpleTable/table.htm” for the table includes a placeholder for the context menu.

```
<script>
    new Table ("${name.id}",${size.jsInt},...,${contextMenu.
jsArr},...)
</script>
```

At runtime, the placeholder is replaced by the Javascript array. The labels and confirmation messages are localized.

```
<script>
    new Table ("roles",15,...,
    ['contextMenuRoles.do',
        ['Entry','E'],['Display summary','summary'],['Modify data',
'modify'],[],
        ['Assign privileges','assignPrivileges'],
        ['Assign users','assignUsers'],['Remove users','
unassignUsers'],[],
        ['Run report','runReport'],
        ['Selected entries','S'],['Delete','delete','Delete the
selected roles?',1,0],
        ['List','L'],['Export','export']],...)
</script>
```

The first array item denotes the Struts action that handles requests initiated via the context menu, in this case “contextMenuRoles.do”. Usually, this is just the context menu name, without leading dot but with extension “.do”.

Each other array item comprises the label of a menu section header or item, a forward and, optionally, a confirmation message text, the index of the default button of the confirmation box, and the due date enabled flag. Some special forwards (E, S and L) are used to denote the section headers. Empty arrays denote separators.

11.1.3.5. Context Menu Display

The browser creates the new table. On context menu activation, it creates the corresponding DOM elements, displays the menu, and handles events like item selection. The context menu layout is affected by several CSS classes defined in /resources/build/style/<fontSize>/styles.css.

11.1.3.6. Requests on Context Menu Selection

On selection of a context menu item, the browser sends a request for the action configured for that item in menu-defs.xml. The request parameters depend on the subset of entries the item is applied to.

- **Entry** – The request includes the following parameters:
- **dn** – The DN of the highlighted entry.
- **forward** – The item forward, the value of the item's label attribute in the menu configuration.
- **startAction** – With value "true". The parameter is evaluated by the JSP checkSession.jsp and initiates a session clean-up.
- **Selected Entries** – The request includes the following parameters:
- **forward** – The item forward, the value of the item's label attribute in the menu configuration.
- **<selectedItems>** – The indexes of the selected items. The exact parameter name depends as usual on the table definition.
- **startAction** – With value "false". The parameter is evaluated by the JSP checkSession.jsp. In this case, it prevents a session clean-up since the table is stored in session-scope and used to handle the request.
- **List** – The request includes the following parameters:
- **forward** – The item forward, the value of the item's label attribute in the menu configuration.
- **startAction** – With value "false". The parameter is evaluated by the JSP checkSession.jsp. In this case, it prevents a session clean-up since the table is stored in session-scope and used to handle the request.

11.1.3.7. Context Menu Handler Actions

Struts actions handling requests initiated by context menus usually forward to the action JSP "/WEB-INF/jsp/controller/tasks/contextMenu.jsp" and define the name of the session-scoped variable that contains the selected entry.

The JSP simply assigns the value of request parameter "dn" to that variable, and forwards to the value of request parameter "forward". Therefore, the labels used in the context menu configuration must match the action forwards of the Struts action.

The following sample shows the action handler for the roles context menu:

```
<action path="/contextMenuRoles"
        type="com.siemens.webMgr.controller.action.JSPAction"
        parameter="jspPage:/WEB-
INF/jsp/controller/tasks/contextMenu.jsp;
        selectedObject:com.siemens.webMgr.selectedRole">
    <forward name="assignPrivileges"
path="/assignRolePrivileges.do"/>
    <forward name="assignUsers"         path="/usersToRole.do"/>
    <forward name="delete"
path="/deleteSelectedRoles.do"/>
    <forward name="export”              path="/exportRoleList.do"/>
    <forward name="modify"              path="/editRoleData.do"/>
    <forward name="runReport"           path="/runReportForRole.do"/>
    <forward name="summary"             path="/showRoleData.do"/>
    <forward name="unassignUsers"       path="/usersFromRole.do"/>
</action>
```

12. Form and List Frames

12.1. Overview

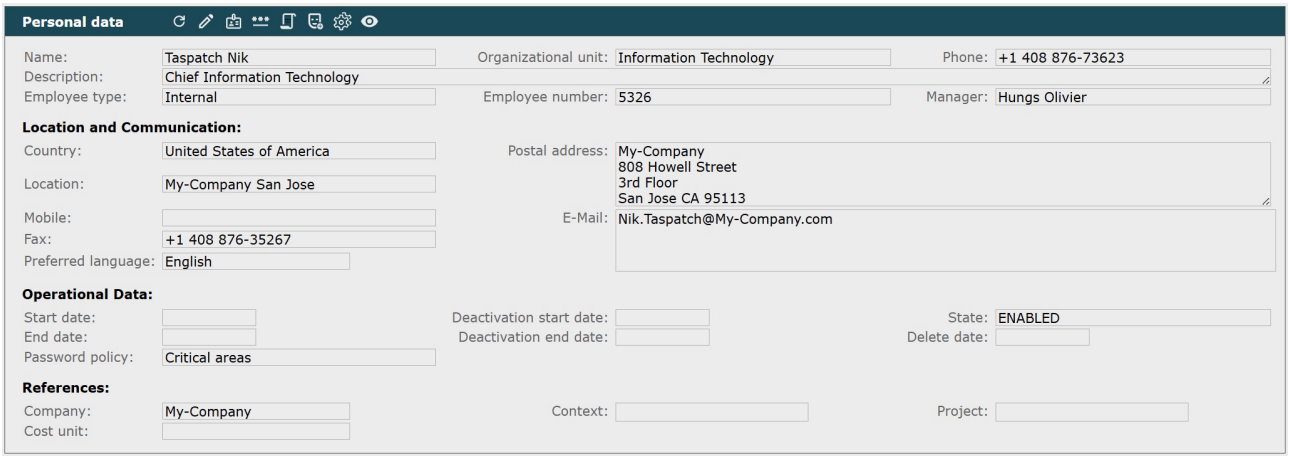
Web Center display frames around some forms and lists. A form frame must have a title and may have a subtitle and a toolbar. A list frame may have a toolbar and a page size selector.

The tools in a toolbar serve as shortcuts to methods in the main menu for forms and in the context menu for lists. Each tool is visualized by an icon with a tooltip and an optional label.

The Web Center standard application shows frames around most summary and modify forms. You can have frames around other forms as well, but note that some pages display more than one form.

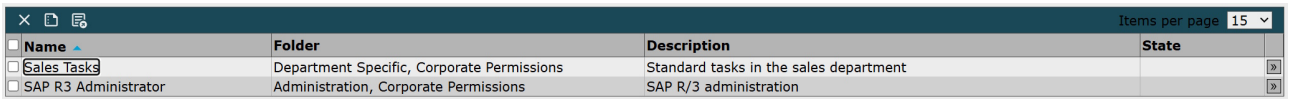
For example, the top row with attributes identifying a user (usually name, organizational unit and phone) forms a separate form on many user pages. If you assign a frame to the main form, the top row will be displayed outside of the frame.

A Sample Form Frame



The frame title is “Personal data”.The tools have icons and a tooltip text.

A Sample List Frame



The frame displays a toolbar with icons only and a page size selector.

12.2. Titles

A form frame must have a title and may have a subtitle. For example, the title for the user overview form is “User data” while the user modification form has title “User data” and subtitle “Edit mode”. Title and subtitle are separated by a hyphen.

12.2.1. Defining Titles

Titles and subtitles are message texts defined per supported language in message file **text.properties**. Custom extensions should be done in custom message file **text_<language>.properties**.

The message key prefix for titles is “form.title.”, the prefix for subtitles is “form.subTitle”.

12.2.1.1. Sample

```
form.subTitle.edit   = Edit mode
form.title.data.self = Personal data
form.title.data.user = User data
```

12.2.2. Assigning Titles

Titles and subtitles are assigned to forms in Tiles configuration files **tiles-defs.xml**.

Assign the relative message keys of title and subtitle to attribute “formTitles” of the Tiles definition displaying the form. Separate title and subtitle by a semicolon.

A form with title but no subtitle

```
<definition name=".overviewUserForm"
            path="/WEB-INF/jsp/view/tiles/form.jsp">
    <put name="formTitles" value="data.user"/>
    …
</definition>
```

The message key for the title is “form.title.data.user”.

12.2.2.1. A form with title and subtitle

```
<definition name=".modifyUserForm"
            path="/WEB-INF/jsp/view/tiles/form.jsp">
    <put name="formTitles" value="data.user:edit"/>
    …
</definition>
```

The message key is “form.title.data.user” for the title and “form.subTitle.edit” for the subtitle.

12.3. Toolbars

12.3.1. Defining Toolbars

Toolbars are defined in menu definition files **menu-defs.xml**.

12.3.1.1. Form Toolbars

Each form toolbar definition has a unique name and two attributes. Attribute “menu” accepts the name of a menu definition. Attribute “items” contains a semicolon-separated list of items from the referenced menu. The toolbar displays the icons in the same order as listed here. Toolbar items are visible only if the referenced menu item is visible.

An item includes the item name, a reference to the corresponding menu item and HTTP request parameters. The item name is used to find the icon and in label keys. The menu item reference defines which action is triggered by the tool and under which conditions the icon is displayed. The HTTP request parameters are transparently sent to the server when the icon has been clicked.

12.3.1.1.1. Sample

```
<definition name=".toolbarUser">
    <put name="menu" value=".menuUserManagement"/>
    <put name="items"
         value="refresh:summary:history=false;modify;resetPassword;

assignPrivileges;copyPrivileges;showSubscriptionStatus;
                taskList;runReport;move;select;lastSelection"/>
</definition>
```

12.3.1.2. List Toolbars

Each list toolbar definition has a unique name and two attributes. Attribute “menu” accepts the name of a context menu definition. Attribute “items” contains a semicolon-separated list of items from the referenced context menu. The toolbar displays the icons in the same order as listed here. Toolbar items are visible only if the referenced context menu item is visible.

List tools are applied to all elements in the list. Therefore, only items in the list section of the context menu will work in toolbars out of the box.

Other items require implementation extensions to be applicable to entire lists. An extension item is added to the item list with extension name, followed by a semicolon and the context menu item name. The extension item replaces the context menu item when searching for icons, labels and tooltips. Also, the respective context menu action must have a forward with the extension item name that is mapped to the Struts action implementing the extended functionality.

Web Center provides a single extension for deleting all entries in a list.

12.3.1.2.1. Sample

```
<definition name=".toolbarRoles">
    <put name="menu" value=".contextMenuRoles"/>
    <put name="items"
value="deleteList:delete;runReportOnList;export"/>
</definition>
```

The sample includes the extension “deleteList”.

12.3.2. Icons

Icons for the standard Web Center are background images 20px wide and 20px high for the medium font size (24px * 24px for large and 16px * 16px for the small font size) defined in CSS style sheets like **styles.css**. Each icon is defined by a CSS class and the image is stored as a SVG data string in the data URL parameter of the "background-image" statement:

```
.tool_create,
.tool_createDirect {
    /* { background-
image:url(../../../../resources/images/svg_backup/add.svg) } */
    background-image:
        url("data:image/svg+xml;utf8, \
            <svg xmlns='http://www.w3.org/2000/svg' height='20'
                    viewBox='0 96 960 960' width='20'> \
                <path fill='%23ECECED' d='M444
                    816V612H240v-72h204V336h72v204h204v72H516v204h-
72Z'/> \
            </svg>")
}
```

The CSS classes "tool_create" and "tool_createDirect" in the sample above define the icon for the toolbar actions "Create" and "CreateDirect". The "background-image" statement in the comment define the storage location relative to the CSS file to a SVG containing this icon. This icon canb be used as a reference.

12.3.3. Labels and Tooltips

Labels and tooltips are defined per supported language in message file **text.properties**. Custom extensions should be done in custom message file **text_<language>.properties**.

The message key for the label of a toolbar item is “toolbar.<itemName>”. The message key for the tooltip of a toolbar item is “toolbar.<itemName>.tooltip”. The default tooltip is identical to the label.

12.3.3.1. Sample

```
toolbar.create              = Create
toolbar.delete              = Delete
toolbar.deleteList          = Delete
toolbar.deleteList.tooltip = Delete list
```

12.3.4. Assigning Toolbars

Each toolbar is assigned to one or more forms or lists in Tiles configuration files **tiles-defs.xml**.

12.3.4.1. Form Toolbars

The toolbar is assigned to form attribute “formToolbar”. Note that a form frame (and thereby a toolbar) is displayed only if a title is defined in attribute “formTitles”.

12.3.4.1.1. Sample

```
<definition name=".overviewUserForm"
            path="/WEB-INF/jsp/view/tiles/form.jsp">
    <put name="formTitles"  value="data.user"/>
    <put name="formToolbar" value=".toolbarUser"/>
    ...
</definition>
```

12.3.4.2. List Toolbars

The toolbar is assigned to list attribute “listToolbar”. Since lists are updated asynchronously via AJAX, you must assign the toolbar to both definitions the one for synchronous requests and the one for update requests.

12.3.4.2.1. Sample

```
<definition name=".rolesForm"
            path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="form"        value="roleListForm"/>
    <put name="listToolbar" value=".toolbarRoles"/>
    ...
```

```
</definition>
<definition name=".updateRoles"
            path="/WEB-INF/jsp/view/tiles/updateList.jsp">
    <put name="form"        value="roleListForm"/>
    <put name="listToolbar" value=".toolbarRoles"/>
    ...
</definition>
```

12.4. Page Size Selectors

A list toolbar may include a page size selector. The end user can select a size from the list, and the list is then redisplayed with the new page size. The user can also choose to display all entries on a single page. The selected value is not stored; each subsequent list is displayed with its default page size.

Each selector has a unique name. The only predefined selector has name “default”.

The label for selectors has message key “list.itemsPerPage”.

12.4.1. Defining Page Size Selectors

A selector is defined in file **webCenter.properties**. The property name is “listPageSizeItems.<selectorName>”. The property value is a Javascript array with the page sizes. A specific page size is ‘all’ to display all entries on a single page.Note that on each display of a list its initial page size is added to the page size items.

12.4.1.1. Sample

The predefined default selector is

```
listPageSizeItems.default = [25,50,100,'all']
```

You can define custom selectors like

```
listPageSizeItems.custom = [7,19,37,71,'all']
```

12.4.2. Assigning Page Size Selectors

The selector name is assigned to list attribute “listPageSizeItems”. Since lists are updated asynchronously via AJAX, you must assign the name to both definitions the one for synchronous requests and the one for update requests.

12.4.2.1. Sample

```
<definition name=".rolesForm"
            path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="form"              value="roleListForm"/>
    <put name="listPageSizeItems" value="default"/>
    …
</definition>
<definition name=".updateRoles"
            path="/WEB-INF/jsp/view/tiles/updateList.jsp">
    <put name="form"              value="roleListForm"/>
    <put name="listPageSizeItems" value="default"/>
    …
</definition>
```

To assign the above custom selector to the roles list, change the definitions to

```
    <put name="listPageSizeItems" value="custom"/>
```

12.5. Global Options

File **webCenter.properties** contains some configuration options that apply to all frames.

12.5.1. Form Frames

- **formShowFrames** – Whether to display list frames. If frames are disabled, titles and toolbars will not be visible.
- **formShowToolbars** – Whether to display list toolbars.
- **formShowToolbarBorders** – Whether to display a border around each tool.
- **formShowToolbarLabels** – Whether to display labels alongside the tool icons.
- **formShowToolLabelPrefix** – Whether to generate tool label prefixes for accessibility tools. The option accepts the following values:
  - **false** – Don't display any prefix.
  - **true** – Display a toolbar specific prefix; the prefix must be defined in message file **text.properties** with key "<toolbarName>.toolLabelPrefix", for example

    ```
    toolbarRole.toolLabelPrefix = Role tool
    ```

  - **default** – Display a generic prefix; the prefix must be defined in file **text.properties** with key "toolbarDefault.toolLabelPrefix".

- **formShowToolLabelPrefix –** Whether to generate tool label prefixes for accessibility tools.

12.5.1.1. Sample

The default options are

```
formShowFrames          = true
formShowToolbars        = true
formShowToolbarBorders  = false
formShowToolbarLabels   = false
formShowToolLabelPrefix = default
```

12.5.2. List Frames

- **listShowFrames –** Whether to display list frames. If frames are disabled, toolbars and page size selectors will not be visible.
- **listShowToolbars –** Whether to display list toolbars.
- **listShowToolbarBorders –** Whether to display a border around each tool.
- **listShowToolbarLabels –** Whether to display labels alongside the tool icons.
- **listShowToolLabelPrefix –** Whether to generate tool label prefixes for accessibility tools. The option accepts the following values:
  - **false –** Don't display any prefix.
  - **true –** Display a toolbar specific prefix; the prefix must be defined in message file **text.properties** with key "<toolbarName>.toolLabelPrefix".
  - **default –** Display a generic prefix; the prefix must be defined in file **text.properties** with key "toolbarListDefault.toolLabelPrefix".
- **listPageSizeItems.<name> –** Defines the items for a page size selector items, a Javascript array. Each array item is either a Javascript number (a page size), or the string "all" (display all items on a single page). Each selector has a name which is referenced in the tile definitions of the item lists. The standard Web Center applications use the name "default".

12.5.2.1. Sample

The default options are

```
listShowFrames          = true
listShowToolbars        = true
listShowToolbarBorders  = false
listShowToolbarLabels   = false
listShowToolLabelPrefix = default
```

```
listPageSizeItems.default = [25,50,100,'all']
```

13. Language and Font Size Chooser

Each Web Center page displays two controls to let users choose their preferred language and font size. This chapter shows you how to hide one or both controls, adapt their value ranges and define their default values.

13.1. Language Chooser

13.1.1. Valid Languages

A valid language is the name of a folder in the directory **WEB-INF/classes/resources/languages**.The folder must contain the message file **text.properties**.

Supported formats are *<language>* and *<language>_<country>* with language and country as used by the Java class **java.util.Locale**, for example **de**, **de_CH**, **en**, **en_US**.

The standard Web Center applications support the languages **de** and **en**.

13.1.2. Configuring the Language Chooser

Labels and values for the language chooser are defined per supported language in file **WEB-INF/classes/resources/<language>/text.properties**.

The message key for the chooser's label is **application.option.language**:

```
application.option.language = Language:
```

or, in the German message file:

```
application.option.language = Sprache:
```

Labels and values for the supported languages are defined by the message with key **application.option.languages**. The message text lists each language and its label.

```
application.option.languages = en:English;de:Deutsch
```

13.1.3. Defining the Default Language

The default language is defined in file **WEB-INF/web.xml** by configuration parameter **javax.servlet.jsp.jstl.fmt.fallbackLocale**:

```
javax.servlet.jsp.jstl.fmt.fallbackLocale = en
```

The hard-coded fallback language is **en**. The value must be one of the valid languages.

Once a user has selected a different language it is stored as part of the login cookie (provided the user accepts the cookie). The cookie is permanently stored on the client for a maximum of 30 days. The cookie language takes precedence over the default language. Note that you can disable login cookies and define a different maximum age in **webCenter.properties**.

The selected language is also stored in a session variable with name **com.siemens.webMgr.language**. This assures that the language applies to all requests for the current session even if the cookie is rejected.

13.1.4. Removing the Chooser

To remove the chooser from your application assign the value “false” to parameter **headerOptionsLanguage** in file **WEB-INF/config/webCenter.properties**:

```
headerOptionsLanguage = false
```

13.2. Font Size Chooser

13.2.1. Valid Font Size Names

A valid font size name is the name of a folder in the directory **resources/build/styles**.The folder must contain the stylesheet **styles.css**.

The font size names for the standard Web Center applications are **small**, **medium** and **large**.

13.2.2. Available Font Size Names

The list of available font size names must be assigned to configuration parameter **availableStyles** in file **WEB-INF/config/webCenter.properties**:

```
availableStyles = large,medium,small
```

The list is evaluated on requests to change the font size. If you remove a font size from the list requests to set the font size to that value are ignored even if the size is selectable from the font size chooser.

13.2.3. Configuring the Font Size Chooser

Labels and values for the font size chooser are defined per supported font size in file **WEB-INF/classes/resources/<language>/text.properties**.

The message key for the chooser’s label is **application.option.fontSize**:

```
application.option.fontSize = Font size:
```

or, in the German message file:

```
application.option.fontSize = Schriftgröße:
```

Labels and values for the supported font sizes are defined by the message with key **application.option.languages**. The message text lists each font size and its label.

```
application.option.fontSizes = small:Small;medium:Medium;large:Large
```

13.2.4. Defining the Default Font Size

The default font size is defined by configuration parameter **defaultStyle** in file **WEB-INF/config/webCenter.properties**:

```
defaultStyle = medium
```

The hard-coded fallback style is **medium**.

Once a user has selected a different font size it is stored as part of the login cookie (provided the user accepts the cookie). The cookie is permanently stored on the client for a maximum of 30 days. The cookie style takes precedence over the default style. Note that you can disable login cookies and define a different maximum age in **webCenter.properties**.

The selected style is also stored in a session variable with name **com.siemens.webMgr.style**. This assures that the style applies to all requests for the current session even if the cookie is rejected.

13.2.5. Removing the Chooser

To remove the chooser from your application assign the value “false” to parameter **headerOptionsFontSize** in file **WEB-INF/config/webCenter.properties**:

```
headerOptionsFontSize = false
```

13.3. Removing Both Choosers

The recommended way to remove both the language chooser and the font size chooser from an application is to assign the value “false” to parameter **headerOptions** in file **WEB-INF/config/webCenter.properties**:

```
headerOptions = false
```

13.4. Reset Points

A user can choose another language or style on each Web Center page. That raises the problem of which page to revert to after the change. One cannot simply resend the previous request since this could mean redoing a modification or an assignment, or since the request parameters are no longer available or the session state has changed in between.

If the navigation history is enabled and not empty, the user will be redirected to the most recent page in the history.

If the navigation history is disabled or still empty, the user will be redirected to the last reset point. A reset point is a Struts action to which the application can safely redirect to after having processed a request to change an option. A reset point action must not perform any changes in the database, and should not rely on a specific session state. Web Center keeps track of the last reset point visited per session, and redirects to that action after option changes.

An action is declared a reset point by adding a parameter with name **resetPoint** and value “true” to the action definition in a **struts-config.xml** file, for example:

```
<!-- Action to list users -->
<action path="/getUsers"
    type="com.siemens.webMgr.controller.action.JSPAction"
    name="userListForm"
    parameter="jspPage:/WEB-INF/jsp/controller/core/listObjects.jsp;
    resetPoint:true;
    ...>/
```

14. Utility Bar

The utility bar is the horizontal area between header and content area. It displays the navigation history, a quick search control and a link to an advanced search page.



The standard Web Center application displays a control to search for users, and a link that leads to the user search panel. You can reconfigure the control to search for another type of object, like roles. And you can let the link point to another search panel.

Control and link are the same on each page of an application; they do not vary with the content of the main area.

The visibility of the utility bar may be controlled by a menu policy. For example, the standard utility bar is visible to users that are allowed to search for users via the main menu bar.

The visibility of the utility bar and its components is also controlled by some global properties in configuration file **webCenter.properties**.

14.1. Utility Bar Definition

The utility bar is defined in **config/identity/tiles-defs.xml**. The definition supports the following attributes:

- **navigationHistory** – The Tiles definition name of the navigation history.
- advancedSearch – The Tiles definition name of the advanced search link.
- **quickSearch** – The Tiles definition name of the quick search link.
- **menuPolicyKey** – The key of the menu policy controlling the visibility of the utility bar.
- **menuPolicyItem** – The item of the menu policy controlling the visibility of the utility bar.

The Web Center provides quick search and advanced search definitions for users, roles, permissions and groups.

Sample

```
<definition name=".utilityBar"
            path="/WEB-INF/jsp/view/tiles/utilityBar/utilityBar.jsp">
    <put name="navigationHistory"  value=".navigationHistory"/>
    <put name="advancedSearch"     value=".usersAdvancedSearch"/>
    <put name="quickSearch"        value=".usersQuickSearch"/>
    <put name="menuPolicyKey"      value="UserMgt"/>
    <put name="menuPolicyItem"     value="select"/>
</definition>
```

14.2. Navigation History

The navigation history control allows the user return back to some pages that were displayed previously. The control displays just meaningful pages and is highly recommended to be used instead of built-in browser back navigation.

The navigation history control is defined in a Tiles definition file **tiles-defs.xml**.

The definition doesn't support any attributes.

Sample

```
<definition name=".navigationHistory"
    path="/WEB-INF/jsp/view/tiles/utilityBar/navigationHistory.jsp"/>
```

14.2.1. Maximum Number of Items

The maximum number of items in the navigation history list is configured in **WEB-INF/config/webCenter.properties** under key navigationHistory.maxItems.

```
navigationHistory.maxItems = <number>
```

Sample

```
navigationHistory.maxItems = 15
```

14.2.2. Configuring Tooltips and Labels

Tooltips and labels for the navigation history are configured in message file **text.properties**.

A general prefix for navigation history components to be used by accessibility tools to distinguish them from other page components:

```
navigationHistory.label          = History
```

The tooltips for the back button, the selection list and the forward button:

```
navigationHistory.label.back     = Go to previous page
navigationHistory.label.forward  = Go to next page
navigationHistory.label.select   = Visited pages
```

The history item labels are automatically derived from the menu whenever possible. Other item labels must be explicitly configured:

```
navigationHistory.<actionName> = <item display text>
```

The action name is the name of the Struts action referenced by the history item. The message supports expressions that reference other message keys, as well as argument expressions similar to the main menu.Some examples are:

```
navigationHistory.listAccounts   = List accounts
navigationHistory.home           = ${home.home}
navigationHistory.resetPassword = ${users.resetPassword}: {30dn}
navigationHistory.runReport      = Run report: {30s}
```

14.3. Quick Search

A quick search control is defined in a Tiles definition file **tiles-defs.xml**. The definition supports the following attributes:

- **action –** The Struts action to be invoked to perform the search operations (without leading slash).
- **objectClass –** The object class for the search filter.
- **operand –** The search filter operand.Supported values are “beginsWith”, “contains”, “endsWith” and “isPresent”.
- **attribute –** The name of the filter attribute.You can specify a comma-separated list of attribute names for an “or”-filter.
- **type –** The type of objects to be searched, like “users” or “roles”. Is used only to display type specific prompts, labels and so on.

Sample

The base quick search definition in **WEB-INF/config/identity/tiles-defs.xml** defines the JSP displaying the control and the default filter operand:

```
<definition name=".quickSearch"
            path="/WEB-
INF/jsp/view/tiles/utilityBar/quickSearch.jsp">
    <put name="operand" value="beginsWith"/>
</definition>
```

The user quick search definition in **WEB-INF/config/users/tiles-defs.xml** is derived from the base definition. Therefore, it inherits filter operand “beginsWith”.

```
<definition name=".usersQuickSearch" extends=".quickSearch">
    <put name="action"      value="getUsers.do"/>
    <put name="attribute"   value="cn"/>
    <put name="objectClass" value="dxrUser"/>
    <put name="type"        value="users"/>
</definition>
```

The user quick search finds all users whose common name starts with the entered value.

If you change the definition to

```
    <put name="attribute"   value="sn,gn"/>
```

a quick search will return users whose surname or given name starts with the input value.

14.3.1. Configuring Tooltips and Labels

Tooltips and labels for the quicksearch control are configured in message file **text.properties**.

A general prefix for quicksearch components to be used by accessibility tools to distinguish them from other page components:

```
utility.quickSearch.label = Quicksearch
```

The label for the input field is not displayed but used by accessibility tools to create a distinct and meaningful name for the field:

```
utility.quickSearch.label.roles = Role name
utility.quickSearch.label.users = User
```

The initial text for the input field:

```
utility.quickSearch.prompt.roles = Enter role name
utility.quickSearch.prompt.users = Enter user name
```

The tooltip for the search button:

```
utility.quickSearch.title.roles = Search roles
```

```
utility.quickSearch.title.users = Search users
```

14.4. Advanced Search

The advanced search link is defined in a Tiles definition file **tiles-defs.xml**. The definition supports the following attributes:

- **action –** The Struts action displaying the advanced search page (without leading slash).
- **label –** The message key for the text for the link.

Samples

The base advanced search definition in **WEB-INF/config/identity/tiles-defs.xml** defines the JSP displaying the control and the default label:

```
<definition name=".advancedSearch"
            path="/WEB-
INF/jsp/view/tiles/utilityBar/advancedSearch.jsp">
    <put name="label" value="utility.advancedSearch.label"/>
</definition>
```

The user advanced search definition in **WEB-INF/config/users/tiles-defs.xml** is derived from the base definition. Therefore, it inherits the label.

```
<definition name=".usersAdvancedSearch" extends=".advancedSearch">
    <put name="action" value="getUsers.do"/>
</definition>
```

14.4.1. Configuring the Link Text

The text for the advanced search link is configured in message file **text.properties**:

```
utility.advancedSearch.label = Advanced search
```

14.5. Global Options

The following options in file **webCenter.properties** control the visibility of the utility bar and its components:

- **utilityBar -** Whether to display the utility bar.Default: true.
- **utilityNavigationHistory –** Whether to display the navigation history control.Default:

true.

- **utilityQuickSearch –** Whether to display the quick search control.Default: true.
- **utilityAdvancedSearch –** Whether to display the advanced search link.Default: true.

Sample

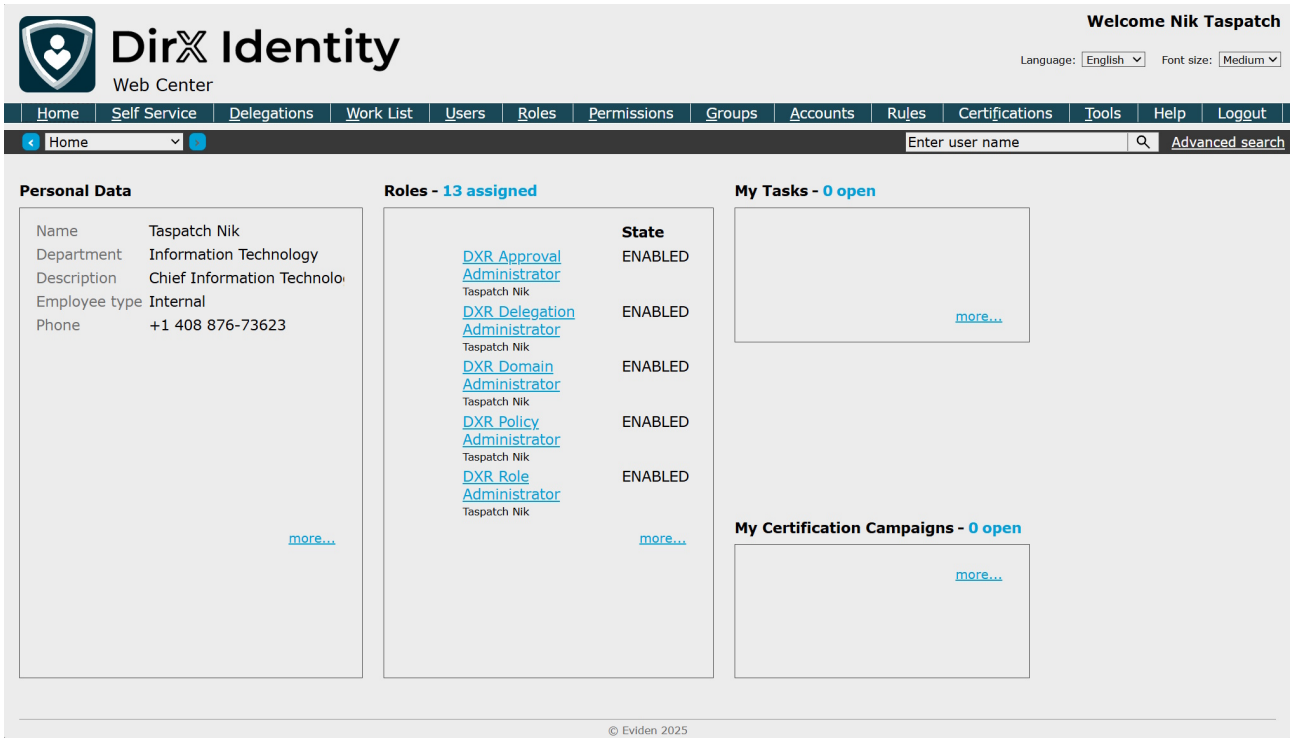
```
utilityBar                = true
utilityNavigationHistory  = true
utilityQuickSearch        = true
utilityAdvancedSearch     = true
```

15. Home Page

The Web Center Home page is displayed as first page after a login. It can also be invoked later on from a button in the user's main menu bar.

A home page is a panel divided into a number of sections. Each section displays a plug-in page with a title, some content and a "more" link. Each plug-in is implemented by a separate Struts action and loaded asynchronously into the home page.

Web Center supports two types of plug-ins: property pages to display attributes of an object, and lists to display entry lists. The default home page includes a property plug-in with some attributes of the logged-in user and two list plug-ins with the roles assigned to the logged-in user, and his open tasks.



For performance reasons, each plug-in shows only a limited amount of data. The "more" link provides a quick link to the detailed property page or complete list. The title of plug-in lists also displays the total number of entries found.

Web Center provides plug-ins for

- A property page with personal data of the logged-in user.
- The roles assigned to the logged-in user.
- The permissions assigned to the logged-in user.
- The groups assigned to the logged-in user.
- The accounts assigned to the logged-in user.
- The logged-in user's task list.

You can customize the provided plug-ins, and you can also define your own plug-ins.

15.1. Defining the Home Menu

The home menu is defined in file **WEB-INF/config/identity/menu-defs.xml**

```
<definition name=".menuHome" extends=".menu">
    <put name="msgPrefix" value="home"/>
    <put name="titleItem" value="home:/home.do:loginDN"/>
</definition>
```

The action invoked on clicking menu button “home” is “/home.do”. The action displays the home page.

The home menu is assigned as first menu to the default menu bar:

```
<definition name=".defaultMenubar">
<put name="items"
     value=".menuHome;sep;.menuSelfService;.menuDelegation;…"/>
</definition>
```

15.2. Displaying the Home Page after Login

By default, the home page is displayed after a user has logged-in. This is achieved by

- Adding the menu item as first item to the start action list in file **WEB-INF/config/identity/menu-defs.xml**:

  ```
  <definition name=".startActions">
      <put name="items"
           value=".menuHome:home;.menuUserManagement:select;…"/>
  </definition>
  ```

- Adding a forward to action “/getStartAction” in file **WEB-INF/config/identity/struts-config.xml**:

  ```
  <action path="/getStartAction" ...>
      <forward name=".menuHome:home" path="/home.do" redirect=
  "true"/>
      …
  </action>
  ```

15.3. Defining the Home Action

The home action is defined in file **WEB-INF/config/identity/struts-config.xml**. It just forwards to the Tiles definition ".home".

```
<action path="/home"
        type="com.siemens.webMgr.controller.action.JSPAction"
        parameter="resetPoint:true;
                   history:true">
    <forward name="next" path=".home"/>
</action>
```

The "history" parameter is set to "true" in order to get home page requests added to the navigation history.

15.4. Defining the Home Page

The home page is defined in file **WEB-INF/config/identity/tiles-defs.xml**. It displays just a form content but no header text.

```
<definition name=".home" extends=".base">
    <put name="formContent" value=".homeForm"/>
</definition>
```

The form content is implemented by a special JSP. The JSP displays a page with a couple of frames that asynchronously load the plug-ins:

```
<definition name=".homeForm"
            path="/WEB-INF/jsp/view/tiles/plugInPage.jsp">
    <put name="pageId"  value="home"/>
    <put name="plugIns" value="…"/>
</definition>
```

The value of attribute "plugIns" is a semicolon-separated list of plug-in definitions.

A plug-in definition includes up to seven components, separated by colons.

- The message key of the plug-in title. For list plug-ins, the key is also used as prefix for the message keys used to display the total number of entries found. The entire plug-in title is then composed of:
  - The plug-in title with message key "<titleKey>", for example "Roles".

    A delimiter with message key "<titleKey>.delim", for example "-".

    - A prefix for the number of entries found with key “<titleKey>.prefix”.
    - The total number of entries found.
    - A suffix for the number of entries found with key “<titleKey>.suffix”, for example “assigned”.
- The plug-in type, either “list” for list plug-ins or “properties” for property-page plug-ins.
- The action displaying the plug-in (without leading slash).
- The optional message key of the text for the “more...” link.
- The optional action to be invoked on clicking the “more...” link. The action may include request parameters.
- An optional comma-separated list of modifiers that control the visibility of the plug-in.
    - Use the predefined modifier “ps” for plug-ins to be visible only if the professional suite is installed.
    - Use the predefined modifier “bs” for plug-ins to be visible only if the business suite is installed.
    - Use modifier “<menu key>:<menu item>” for plug-ins to be visible only if the logged-in user is allowed to execute the corresponding menu item.
    - Any other modifier is ignored.
- A focus priority. The plug-in with the highest priority gets the focus when the page is displayed.

The default plug-in list includes a property-page plug-in and two list plug-ins:

```
value="home.summary:properties:homeSummary.do:
            home.more:showMyUserData.do::4;
       home.groups:list:homeGroups.do:
            home.more:showMyUserData.do?

privilegeAction=showMyUserGroups.do:bs:2;
       home.roles:list:homeRoles.do:
            home.more:showMyUserData.do?

privilegeAction=showMyUserRoles.do:ps:0;
       home.tasks:list:homeTasks.do:
            home.more:showWorklist.do:ps,Worklist.taskList:5"
```

15.5. Property Plug-Ins

A property plug-in displays attributes of an entry. It supports one attribute per line. Each attribute has a label and one or more values. A value is truncated if it exceeds a maximum size. The complete value can be viewed as tooltip or by selecting the value and scrolling to

its end.

The definition of a property plug-in consists of a form bean, a Struts action and a Tiles definition. The samples below can be found in folder **WEB-INF/config/users/home**.

15.5.1. Form Definition

The form bean lists the properties to be displayed. Special renderers are provided for labels and value. Define each value as read-only and on a separate line (y="+1").

15.5.1.1. Sample

```
<form-bean name="homeSummaryForm">
  <form-property-group name="homeSummary" groupRenderer="fieldSet"
                       label="home.summary">
    <form-property name="$displayName"
                   type="java.lang.String" readonly="true"
             labelRenderer="plugInLabel" fieldRenderer="plugInText"
                   y="+1"/>
    <form-property name="dxrOULink@$displayName"
                   type="java.lang.String" readonly="true"
                label="ldap.attribute.ou"
                   labelRenderer="plugInLabel"
fieldRenderer="plugInText"
                   y="+1"/>
    <form-property name="description"
                   type="java.lang.String" readonly="true"
             labelRenderer="plugInLabel" fieldRenderer="plugInText"
                   y="+1"/>
    <form-property name="employeeType"
                   type="java.lang.String" readonly="true"
             labelRenderer="plugInLabel" fieldRenderer="plugInText"
                   y="+1"/>
    <form-property name="telephoneNumber"
                   type="java.lang.String" readonly="true"
             labelRenderer="plugInLabel" fieldRenderer="plugInText"
                   y="+1"/>
  </form-property-group>
</form-bean>
```

Note that the form properties are enclosed in a form property group with field renderer "fieldSet". The renderer has no visible effect. It just defines a general prefix (attribute "label") that is used by accessibility tools to generate distinct names for the enclosed form

properties.

15.5.2. Struts Action

The Struts action references the form bean of the property page plug-in and defines the bean scope; usually, scope “request” will do.

Parameter “jspPage” defines the JSP which reads the attributes from the database and assigns them to the form bean. JSP page **showData.jsp** evaluates the following additional parameters:

- **object –** The DN of the entry whose attributes are to be displayed.

Action forward “tile” is mapped to the Tiles definition displaying the form.

15.5.2.1. Sample

```
<action path="/homeSummary"
        type="com.siemens.webMgr.controller.action.JSPAction"
        name="homeSummaryForm"
        scope="request"
        parameter="jspPage:/WEB-INF/jsp/controller/core/showData.jsp;

object:${sessionScope['com.siemens.webMgr.loginDN']}">
    <forward name="tile" path=".homeSummary"/>
</action>
```

15.5.3. Tiles Definition

The Tiles definition is derived from definition “.fragment” since it doesn’t generate a complete HTML page but just an HTML fragment. Attribute “formContent” refers to the Tiles definition generating the form. Since a plug-in usually doesn’t include a text header, attribute “textFile” is left unspecified.

The form is as usual generated by JSP file **form.jsp**. Attribute”action” references the action the form bean is assigned to. Attribute “styleClass” defines a CSS class which sets form margin and width.

15.5.3.1. Sample

```
<definition name=".homeSummary" extends=".fragment">
    <put name="formContent" value=".homeSummaryForm"/>
</definition>
<definition name=".homeSummaryForm"
            path="/WEB-INF/jsp/view/tiles/form.jsp">
```

```
    <put name="action"     value="/homeSummary.do"/>
    <put name="styleClass" value="plugInPropertyPage"/>
</definition>
```

15.6. List Plug-Ins

List plug-ins support lists with up to three properties per entry. The list has one or two columns. The first and third properties are displayed in one row in the first and second column, respectively. The second property is shown in the first column below the first property.

The first property is required, the others are optional. Headers are supported for the first and third property but not for the second one.

The sort order can be specified but the end user cannot re-sort the list. Default sort order is first property first, then the second and finally the third one.

An icon may be displayed in front of each entry. Clicking the icon or the value of the first property can trigger an action. For example, clicking a role in an “assigned roles” list will display the role’s summary page, while clicking a task in a task list will display the page to perform the task. The action might be available only to users having the access right to perform a corresponding menu item.

The samples below can be found in folder **WEB-INF/config/users/home**.

15.6.1. Form Definition

The form bean accepts just a single child element of type <form-property-list>. The form property list accepts up to three child elements of type <list-property>.

15.6.1.1. Form Property List

The <form-property-list> element supports the following attributes:

fieldRenderer – The reference to a renderer definition to be used for displaying the list. The renderer configuration file “renderers-config.xml” must contain an entry with this identifier. See the section "Renderers Configuration renderers-config.xml" in this chapter for details.

icon – The icon to be displayed alongside each list entry; a file name relative to folder resources/images.

iconTitle – The message key for the icon tooltip.

name – The list name. The name must be unique within the form.

openAction – The Struts action to be invoked when clicking on a list item. The action may be optionally followed by a menu key and item saying that the action is only enabled for users that have the corresponding menu access right. Sample: /openRole.do:RoleMgt.summary.

openDescription – The message key of a textual description of the open action. The description is used by accessibility tools to display or read a unique and meaningful text in order to let users invoke the open action in a different way. The expression {0} in the message text will be replaced with the actual list item name.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the list renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the property list. The property values accept expressions.

size – The number of entries to be displayed in the list. The preferred way is to specify the list size as a parameter of the corresponding Struts action.

sortIndexes – A comma-separated list of column indexes defining the sort order. Default is “0,1,2”.

15.6.1.2. List Property

The <list-property> element supports the following attributes:

cellRenderer – The renderer to be used for displaying the cell content of this column. This attribute must be a valid entry in the renderer configuration file renderers-config.xml.

label – The message key for the list column header. Since the first two properties are displayed in the same column, the label of the first property should describe both properties. The label of the second property is always ignored. Any resolved label ending with “:hidden” is not visible. Note that the label is also used by accessibility tools.

messagePrefix – A prefix for message keys related to this data property.

name – The name of the form property described by this element, most often the LDAP name of an attribute of the objects to be displayed. Required. You can make use of placeholder names such as “$displayName”, which are defined in the object description of the corresponding object.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the list property renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the list property. The property values accept expressions.

type – The fully-qualified Java class name of the field underlying this property, optionally followed by "[]" to indicate that the field is indexed. Required.

whiteSpace – The CSS property value that specifies how to handle white space in the values displayed in this column. Sample values are **normal**, **pre** and **nowrap**.

width – The width of the column. We recommend that you specify the width as a percentage of the total list width, since fixed values can lead to undesirable results in different screen resolutions. The width applies to the first and third list property only,

defining the width of the first and second column, respectively.

15.6.1.3. Samples

15.6.1.3.1. Sample 1

The first example shows a list with a single property and no header.

```
<form-bean name="homePermissionsForm">
    <form-property-list name="assignedUserPermissions"
          openAction="/openPermission.do:PermissionMgt.summary"
          openDescription="home.permissions.openDescription">
        <list-property name="$displayName" width="100%"
                       label="home.permissions.header"/>
    </form-property-list>
</form-bean>
```

Sample 2

The example displays a list with a two properties; the second one is ignored. A header is displayed for the second column.

```
<form-bean name="homeGroupsForm">
    <form-property-list name="assignedGroups"
        openAction="/openGroup.do:GroupMgt.summary"
        openDescription="home.groups.openDescription">
        <list-property name="$displayName" width="90%"
              label="home.groups.header"/>
        <list-property name="skip_1"/>
        <list-property name="assign.displayState"
              label="ldap.attribute.dxrState"
              width="10%" cellRenderer="textState"/>
    </form-property-list>
</form-bean>
```

15.6.1.3.2. Sample 3

The example shows a list with a three properties and an icon. A header is displayed for the second column.

```
<form-bean name="homeTasksForm">
    <form-property-list name="workItems"
```

```
                        openAction="/openTask.do"
                        openDescription="home.tasks.openDescription"
                        icon="task.gif"
iconTitle="home.tasks.iconTitle"
                        sortIndexes="2,0,1">
        <list-property name="actdisplayname" width="90%"
                       label="home.tasks.header"/>
        <list-property name="rdn_true_subject_dn"/>
        <list-property name="actexpirationtime" type="java.util.Date"
                       label="requestwf.worklist.expirationDate"
                       width="10%"/>
    </form-property-list>
</form-bean>
```

15.6.2. Struts Action

The Struts action references the form bean of the list plug-in and defines the bean scope; usually, scope “request” will do.

Parameter “jspPage” defines the JSP which reads the entries from the database and populates the form bean. Web Center provides the JSP page **listEntries.jsp** specifically for list plug-ins. The JSP searches for entries in the database but returns a single page of the total result only, together with the total number of entries found.

The JSP evaluates the following additional parameters:

- **elements –** The name of the form-property-list element of the form bean to be populated. The name must be one of the following items:
  - workItems
  - assignedRoles
  - assignedPermissions
  - assignedGroups
  - assignedAccounts
- **indexOfFirstEntry –** The index of the first entry in the page. The very first entry has index 1. Default: 1.
- **indexOfLastEntry –** The index of the last entry in the page. The very first entry has index 1. Default: 5.
- **sortAttributes –** Attributes for server-side sorting; a comma-separated list of LDAP attribute names. Each name may be followed by “:r” to indicate reverse ordering. Server-side sorting is supported for workItems only. For example, “dxrStartDate:r,dxrExpirationDate” sorts for start date in reversed order first, expiration data in regular order second.

Action forward “tile” is mapped to the Tiles definition displaying the form.

15.6.2.1. Sample

Usually the only attribute you might want to change for one of the standard list plug-ins is the index of the last entry to be displayed. To make adaptation easier, the index takes it value from an option in file **webCenter.properties**. Change the option there, and leave the Struts action definition untouched.

```
<action path="/homeTasks"
        type="com.siemens.webMgr.controller.action.JSPAction"
        name="homeTasksForm"
        scope="request"
        parameter="jspPage:/WEB-
INF/jsp/controller/core/listEntries.jsp;
                   elements:workItems;
                   sortAttributes:dxrExpirationDate;

indexOfLastEntry:${webCenter.homeMaxNumberOfTasks}">
    <forward name="tile" path=".homeTasks"/>
</action>
```

15.6.3. Tiles Definition

The Tiles definition is derived from definition “.fragment” since it doesn’t generate a complete HTML page but just an HTML fragment. Attribute “formContent” refers to the Tiles definition generating the form. Since a plug-in usually doesn’t include a text header, attribute “textFile” is left unspecified.

The form is as usual generated by JSP file form.jsp. Attribute”action” references the action the form bean is assigned to. Attribute “styleClass” defines a CSS class which sets the form width.

15.6.3.1. Sample

```
<definition name=".homeTasks" extends=".fragment">
    <put name="formContent" value=".homeTasksForm"/>
</definition>
<definition name=".homeTasksForm" path="/WEB-
INF/jsp/view/tiles/form.jsp">
    <put name="action"     value="/homeTasks.do"/>
    <put name="styleClass" value="plugInList"/>
</definition>
```

15.7. Global Properties

Some properties in file **webCenter.properties** allow setting the maximum number of entries to be displayed on the standard list plug-ins.

```
homeMaxNumberOfAccounts     = 5
homeMaxNumberOfGroups       = 5
homeMaxNumberOfPermissions  = 5
homeMaxNumberOfRoles        = 5
homeMaxNumberOfTasks        = 5
```

16. Sorting Lists

16.1. Sort Attributes

16.1.1. List Attributes

A list is defined via a <form-property> element of type DirectoryEntryBean[]. The list supports two attributes related to sorting.

- **sortable –** A Boolean flag. If set to “false”, the flag disables sorting for all columns in the list. This is used for example for the list of outstanding resource orders on overview pages since sorting would blend rows belonging to different orders.
- **secondarySortColumns –** A comma-separated list of the names or indexes of the columns that serve as secondary sort criteria, for example “0,1” for the first and second column, or “sn,gn” for the columns with name “sn” and “gn”. The default secondary sort criterion is the first column “0”.

16.1.2. Column Attributes

A column is defined via a <data-property> child element of a list. The column supports two attributes related to sorting.

- **sortable –** A Boolean flag. If set to “false”, the flag disables sorting for the column. Note that Web Center doesn’t support sorting for some types of columns.
- **sortIndexes –** A comma-separated list of the indexes of the columns that serve as sort criteria, for example “2,0,1” for the third, first and second column.

16.2. Sort Attribute Evaluation

If a user clicks on the header cell of a table column, the list is sorted according to the following rules (in descending precedence):

- If list attribute “sortable” is set to false, sorting is disabled for each column in the list.
- If column attribute “sortable” is set to false, sorting is disabled for the column.
- If column attribute “sortIndexes” is set, then the list is sorted as specified in the attribute value. Attribute “secondarySortColumns” is ignored.
- The list is sorted by the values in the column. Attribute “secondarySortColumns” is evaluated for secondary sort columns. If left unspecified, the first column serves as secondary sort criterion.

16.3. Samples

Sample 1

The first example doesn’t specify any sort attribute at all. The first columns serves as

secondary sort criterion. This will suit most lists.

```
<form-property name="users">
    <data-property name="sn"/>
    <data-property name="ou"/>
    <data-property name="tn"/>
</form-property>
```

Click on header of column "sn": sort by "sn".

Click on header of column "ou": sort by "ou" first, "sn" second.

Click on header of column "tn": sort by "tn" first, "sn" second.

Sample 2

The next example defines two secondary sort columns. This will suit almost all other lists.

```
<form-property name="users" secondarySortColumns="sn,ou">
    <data-property name="sn"/>
    <data-property name="ou"/>
    <data-property name="tn"/>
    <data-property name="fax"/>
</form-property>
```

Click on header of column "sn": sort by "sn" first, "ou" second.

Click on header of column "ou": sort by "ou" first, "sn" second.

Click on header of column "tn": sort by "tn" first, "sn" second, "ou" last.

Click on header of column "fax": sort by "fax" first, "sn" second, "ou" last.

Sample 3

The next sample defines individual sort orders for the first three columns. The default sort order applies to the 4th column, with the first column as default secondary sort criterion. Sorting is disabled for the last column.

```
<form-property name="users">
    <data-property name="sn" sortIndexes="0,1,2"/>
    <data-property name="ou" sortIndexes="1,2"/>
    <data-property name="tn" sortIndexes="2"/>
    <data-property name="fax"/>
```

```
    <data-property name="gn" sortable="false"/>
</form-property>
```

Click on header of column "sn": sort by "sn" first, "ou" second, "tn" last.

Click on header of column "ou": sort by "ou" first, "tn" second.

Click on header of column "tn": sort by "tn".

Click on header of column "fax": sort by "fax" first, "sn" second.

Click on header of column "gn": is ignored.

Sample 4

The next sample defines individual sort orders for the first two columns. Sorting is disabled for the 3rd and the last column. The default sort order applies to the 4th column, with the second column as secondary sort criterion.

```
<form-property name="users" secondarySortColumns="1">
    <data-property name="sn" sortIndexes="0,1,2"/>
    <data-property name="ou" sortIndexes="1,2"/>
    <data-property name="tn" sortIndexes="2" sortable="false"/>
    <data-property name="fax"/>
    <data-property name="gn" sortable="false"/>
</form-property>
```

Click on header of column "sn": sort by "sn" first, "ou" second, "tn" last.

Click on header of column "ou": sort by "ou" first, "tn" second.

Click on header of column "tn": is ignored.

Click on header of column "fax": sort by "fax" first, "ou" second.

Click on header of column "gn": is ignored.

Sample 5

The last list is not sorted at all.

```
<form-property name="users" sortable="false">
    ...
</form-property>
```

17. Expression Language Extensions

Web Center provides a means to evaluate functions in expressions. The functions may have one parameter only.

Functions can for example be used to:

- Check if an attribute exists for a given entry.
- Check if an attribute has a specific value for a given entry.
- Check if all attribute values of a given entry match a regular expression.

The tags for defining and assigning expression functions are part of the expression tag library which must be declared in JSP files with the include direction

```
<%@ taglib uri="http://www.siemens.com/directory/webManager/expr"
           prefix="expr" %>
```

We recommend using the prefix “expr” but you may choose any other unique prefix.

17.1. Defining Functions

17.1.1. Attribute Exists

The tag “attributeExists” generates a function that checks if an attribute exists.

The following example defines a function with name “isFinished” which checks if the workflow end time is set.

```
<expr:method name="isFinished">
    <expr:attributeExists name="wfEndTime"/>
</expr:method>
```

17.1.2. Attribute Contains

The tag “attributeContains” generates a function that checks if an attribute contains a specific value.

The following example defines a function with name “isFunctionalUser” which checks if one of the values of attribute “objectClass” is “dxrFunctionalUser”. The check is done case-insensitive.

```
<expr:method name="isFunctionalUser">
    <expr:attributeContains
```

```
        name="objectClass"
        value="dxrFunctionalUser"
        caseIgnore="true"/>
</expr:method>
```

17.1.3. Attribute Matches

The tag “attributeMatches” generates a function that checks if one or all values of an attribute match a regular expression.

The following example defines a function with name “isContractorOrSupplier” which checks if one of the values of attribute “employeeType” is “Customer” or “Supplier”. The check is done case-insensitive.

```
<expr:method name="isContractorOrSupplier">
    <expr:attributeMatches
        name="employeeType"
        expression="Contractor|Supplier"
        caseIgnore="true"
        all="false"/>
</expr:method>
```

17.2. Assigning the Functions to a Scoped Variable

We must assign functions to a scoped variable. This provides us with a handle to invoke the functions in expressions later on.We enclose the “method” tags in a “methods” tag and pass the variable’s name and scope as tag attributes.

The following example assigns the three functions defined above to the session-scoped variable with name “apply”.

```
<expr:methods var="apply" scope="session">
    <expr:method name="isFinished">
        ...
    </expr:method>
    <expr:method name="isFunctionalUser">
        ...
    </expr:method>
    <expr:method name="isContractorOrSupplier">
        ...
    </expr:method>
```

```
</expr:methods>
```

We can now invoke the functions for example with “sessionScope.apply.isFinished” or simply “apply.isFunctionalUser” if there’s no variable with name “apply” defined in page or request scope.



The functions access the database which means they need a valid session. That’s why we put them into session scope.

A convenient place to add the code to is the **initSession.jsp** in folder **WEB-INF/jsp/controller/utils**. The JSP is called once for each session.

17.3. Using the Functions

The functions can be used in expressions after they have been assigned to a scoped variable. The function argument must be passed in square brackets []. The argument can be a fixed string enclosed in single quotes, or any scoped variable.

The standard Web Center application uses functions only for menu selectors. But you can use them in expressions anywhere else.

17.3.1. Menu Selectors

The following extract from file **WEB-INF/config/identity/menu-defs.xml** shows how a function is used to define a menu selector with name “notFunctionalUser” that applies if the selected user is not a functional user.

```
<definition name=".menuSelectors">
  <put name="notFunctionalUser"
    value="${not apply.isFunctionalUser
           [sessionScope['com.siemens.webMgr.selectedUser']]}"/>
</definition>
```

The menu selector is used later on in the same file to disable some menu items for functional users.

17.3.2. JSP Pages

This section gives some examples for using functions in expressions within JSP pages.

First, we store the logged-in user and the selected workflow item in page-scoped variables just for convenience:

```
<c:set var="loginUser"
       value="${sessionScope['com.siemens.webMgr.loginDN']}"/>
```

```
<c:set var="selectedWorkflowItem"

value="${sessionScope['com.siemens.webMgr.selectedWorkflowItem']}"/>
```

Now, let's check if the logged-in user is a functional user:

```
<c:choose>
    <c:when test="${apply.isFunctionalUser[loginUser]}">
        <%-- is a functional user --%>
    </c:when>
    <c:otherwise>
        <%-- is not a functional user --%>
    </c:otherwise>
</c:choose>
```

Next, we check if the logged-in user is a contractor or supplier, and store the result in a page-scoped variable:

```
<c:set var="selectedUserIsContractorOrSupplier"
       value="${apply.isContractorOrSupplier[loginUser]}"/>
```

Finally, we check if the selected workflow item is already finished:

```
<c:if test="${apply.isFinished[selectedWorkflowItem]}">
    <%-- is finished --%>
</c:if>
```

18. Accessibility

The W3C Web Accessibility Initiative (WAI) published some guidelines to better support accessibility in HTML applications. The guidelines cover not only plain HTML pages but also so-called rich internet applications (ARIA: “Accessibility for rich internet applications”). In particular, the WAI defined some new attributes (starting with “aria-“) for HTML elements and defined a number of new values for the HTML attribute “role”.

This chapter describes some of the changes to the Web Center code for accessibility. The changes have been tested with JAWS 14.

18.1. Labels

Accessibility tools need unique and meaningful descriptions for each element on a page in order to be able to present the elements properly to the user. The visible element labels are often not sufficient since they don’t contain any information about their context. For example, if a page displays two buttons with label “Roles”, it’s usually clear for a user who is able to see which button is part of the menu bar and which one is part of the privileges tab. If an accessibility tool reads the page or lists all buttons on the page in a separate window, however, a blind user will not be able to distinguish the two buttons without any additional context information.

Web Center sets some HTML attributes defined by the WAI that let accessibility tools generate unique and meaningful labels.

18.1.1. Attribute aria-label

The attribute “aria-label” defines a label for an HTML element for use by accessibility tools. The label is not visible.

Sample

The list pager bar displays just a double left angular quote for the button “Go to first page”. The button has a tooltip but the title attribute is ignored by accessibility tools. Therefore, we add the attribute “aria-label” with the same value as the tooltip. The label is then read or displayed by accessibility tools.

Here is the code from snippet **simpleTable/ListPager.htm**:

```
<td style="padding-right:6px">
    <input type="button"
        id="${name.id}_first" class="labeledButton tablePagerButton"
        value="&laquo;" title="${tooltipFirst.h}"
        aria-label="${tooltipFirst.h}"/>
</td>
```

18.1.2. Attribute aria-labelledby

The attribute “aria-labelledby” also defines a label for an HTML element for use by accessibility tools. In this case, however, the label comprises the values of one or more HTML elements whose HTML ids are listed in “aria-labelledby”. Again, the label is not visible.

Sample

The list pager bar includes an input field for the index of the current page. The user can switch to another page by entering its index. If the input field gets the focus an accessibility tool should read not just the index “2” but “Page 2 of 5” or so.

We assign three HTML ids to attribute “aria-labelledby” of the input field. The first id refers to the cell displaying the text “Page”. The second id refers to the cell containing the input field itself; this adds the page index to the aria label. The third id adds the text “of “ and the total number of pages to the label.

Here is the code from snippet **simpleTable/ListPager.htm**:

```
<td id="${name.id}_label1" …>${labelPage.h}</td>
<td id="${name.id}_label2">
    <input type="text" id="${name}_page"
        class="rwTextField tablePagerInput" value="1"
        aria-labelledby=
          "${name.id}_label1 ${name.id}_label2 ${name.id}_label3"/>
</td>
<td id="${name.id}_label3" ...>
    ${labelOf.h} <span id="${name}_numPages">2</span>
</td>
```

18.1.3. Fieldset

To indicate the context of HTML elements it’s often a good idea to use a common prefix for their ARIA labels. For example, all fields in a search panel should get a label prefix that identifies the search panel, like “Search users”. This also makes sure that the elements are grouped together in lists that can be displayed by accessibility tools, like the list of all editable form elements or of all buttons on a page.

Web Center often encloses associated fields in a fieldset element and assigns an ARIA label to the fieldset which serves as ARIA label prefix for the associated fields. The ARIA label is assigned either via attribute “aria-label” or “aria-labelledby”. The CSS class “aria” makes sure that the fieldset has no visible effects.

Sample

Here is an extract from snippet **searchPanel/searchPanel.htm**:

```
<fieldset aria-label="${searchPanelTitle.h}" class="aria">
   ... search panel fields ...
</fieldset>
```

The message key for the search panel title is by convention “searchPanel.title.<objects>” where “<objects>” is replaced with the attribute of the same name in the tiles definition of the corresponding entry list. The key for the user list is “searchPanel.title.users”, so the ARIA label prefix for the fields in the user search panel is “Search users”.

18.2. ARIA Roles

The WAI defined a number of values for the HTML attribute “role”. The attribute is intended to indicate the semantic of an HTML element.

18.2.1. Presentation

The role “presentation” indicates that an element is used for its presentational effects only but has no semantic meaning. Web Center assigns the attribute to some layout tables that should be ignored by accessibility tools.

Sample

The following layout table for the home page plug-ins is defined in JSP **view/tiles/plugInPage.jsp**:

```
<table class="plugInPage" cellpadding="0" cellspacing="0"
       style="height:100%" role="presentation">
   ... table content ...
</table>
```

18.2.2. Menubar and Toolbar

Web Center assigns the roles “menubar”, “menuitem” and “menu” to HTML elements of the main menu and the roles “toolbar” and “button” to HTML elements of the toolbars for forms and lists. Accessibility tools are therefore able to detect the semantic of the elements. JAWS, for example, will switch its mode of operation so that keyboard events go directly to the menubar or toolbar.

18.3. Read-only Fields in Tab Order

Web Center includes read-only form fields in the tab order by setting their tab index to “0”. This makes it easier for disabled persons to access the field content. You can redefine the tab order in file **webCenter.properties**:

```
# Tab index for read-only form fields
#   -1: exclude from TAB order
#    0: default TAB order

formReadOnlyTabIndex = 0
```

The property is evaluated in snippets for read-only form fields, for example in **textField/roTextField.htm**:

```
<input type="text" id="${id.id}" name="${name.h}"
  class="roTextField" style="${style.h}" value="${value.h}"
  readonly="readonly"
  tabindex="${webCenter.formReadOnlyTabIndex.jsInt}"/>
```



Read-only check boxes are never included in the tab order since in order to be not editable they have to be marked as disabled.

18.4. Editable Date Fields

The values of date fields are by default not directly editable. To change their values you have to select a date in the calendar widget. The widget may be difficult to use by disabled persons.

You can switch to directly editable date fields by changing the properties of the renderers “calendar” and “dueDate” in file **renderers-config.xml**:

```
<renderer id="calendar" type="java.util.Date" ...>
  ...
  <!-- Activate this section for non-editable date input fields -->
  <renderer-property name="formatTooltip"        value=""/>
  <renderer-property name="formInputReadOnly"
      value="readonly=&amp;quot;readonly&amp;quot; tabindex=&amp;quot;-1&amp;quot;"/>
  <renderer-property name="formInputStyleClass"  value=
"roTextField"/>
  <renderer-property name="tableInputStyleClass" value=
"roTextField"/>
  <renderer-property name="tableInputReadOnly"   value="true"/>

  <!-- Activate this section for editable date input fields -->
  <!--
  <renderer-property name="formatTooltip"
```

```
                    value="application.dateFormat.tooltip"/>
  <renderer-property name="formInputReadOnly"    value=""/>
  <renderer-property name="formInputStyleClass"
value="rwTextField"/>
  <renderer-property name="tableInputStyleClass"
value="rwTextField"/>
  <renderer-property name="tableInputReadOnly"   value="false"/>
  -->
</renderer>
```

The renderer properties are evaluated in the corresponding code snippets, for example in **calendar/calendar.htm**:

```
<input type="text" id="${id.id}" name="${name.h}"
    ${formInputReadOnly} class="${formInputStyleClass.h}"
    style="${style.h}" title="${formatTooltip.h}"
    value="${value.h}" onchange="markChanged()"/>
```

18.5. Table Summaries

Table summaries are used by accessibility tools to tell a disabled user what the tables are about. Visual labels like table captions are usually too terse for that purpose.

The message key for a table summary is usually determined by convention but may be overwritten per table by configuration. The following list displays the options with descending priority:

- Value of form-property attribute “summary”
- “list.summary.<form-property name>.<form-bean name>”
- “list.summary.<form-property name>“

Sample: Defining a specific summary key:

```
<form-bean name="userListForm">
    <form-property name="users" summary="users.summary" …
```

The message key is “users.summary”.

Sample: Defining no summary key:

```
<form-bean name="userListForm">
```

```
    <form-property name="users" …
```

The message key is “list.summary.users.userListForm” if a message with that key is defined. Otherwise, the key is “list.summary.users”.

Short (probably too short) default summary texts for the tables used in the default Web Center applications are provided. Here are some summaries from the English **text.properties** file:

```
list.summary.userLinks           = List of selectable users
list.summary.users               = List of users
list.summary.workflowDefinitions = List of available workflows
list.summary.workflowLinks       = List of selectable workflows
```

The summaries are evaluated in the corresponding code snippets, for example in **simpleTable/table.htm**:

```
<table id="${name.id}" width="100%" cellpadding="0" cellspacing="0"
     summary="${summary.h}" class="tableBorder">
```

19. Using the Keyboard

Web Center allows users to navigate the application and manipulate user interface elements by using a keyboard. Some controls are even fully accessible with accessibility tools (like Jaws). This chapter lists the supported keys.

19.1. Fully Accessible User Interface Controls

19.1.1. Main Menu

Navigate to the menu bar with TAB.

19.1.1.1. If the focus is on a menu title:

ESC	Close the menu without selecting a menu item
CR	Execute the menu function (Home, Logout)
	Open menu (otherwise)
SPACE	Execute the menu function (Home, Logout)
	Open menu (otherwise)
LEFT	Close current menu and open next menu
RIGHT	Close current menu and open previous menu
UP	Open menu and highlight first item
DOWN	Open menu and highlight first item

The keys LEFT and RIGHT work in a round-robin fashion.

19.1.1.2. If the focus is on a menu item:

ESC	Close the menu without selecting a menu item
CR	Select the highlighted item and close the menu
SPACE	Select the highlighted item and close the menu
LEFT	Close current menu and open next menu
RIGHT	Close current menu and open previous menu
UP	Highlight the previous menu item
DOWN	Highlight the next menu item

The keys LEFT, RIGHT, UP and DOWN work in a round-robin fashion.

19.1.1.3. Shortcuts

Menu titles can be activated by access keys (ALT <letter> in Internet Explorer and Chrome, SHIFT ALT <letter> in Edge and Firefox).

Items of an open menu can be activated by just typing the corresponding letter (without ALT or SHIFT ALT).

19.1.2. Form and List Toolbar

Navigate to the toolbar with TAB.

19.1.2.1. If the focus is on a tool:

CR	Select the tool
SPACE	Select the tool
LEFT	Move the focus to the next tool
RIGHT	Move the focus to the previous tool

The keys LEFT and RIGHT work in a round-robin fashion.

19.2. Other User Interface Controls

19.2.1. Calendar Widget

ESC	Close the calendar widget without changing the currently selected date

19.2.1.1. If the focus is on a date:

CR	Select the date and close the calendar widget
LEFT	Move the focus to the next date
RIGHT	Move the focus to the previous date
UP	Move the focus to previous date in the same column
DOWN	Move the focus to the next date in the same column

The keys LEFT, RIGHT, UP and DOWN work in a round-robin fashion.

19.2.2. Tree Widget

ESC	Close the tree widget without changing the current selection

19.2.2.1. If an item is highlighted:

CR	Select the highlighted item and close the tree widget
UP	Display and highlight the previous item
DOWN	Display and highlight the next item
PAGE UP	Highlight the first item of the current page if not yet highlighted

CR	Select the highlighted item and close the tree widget
	Highlight the first item of the previous page if the first item of the current page is already highlighted
PAGE DOWN	Highlight the last item of the next page if not yet highlighted
	Highlight the first item of the previous page if the first item of the current page is already highlighted
POS1	Display the first page and highlight the first item
END	Display the last page and highlight the last item
LEFT	Highlight parent if the currently highlighted item is closed
	Close item if the currently highlighted item is open
RIGHT	Open item if the currently highlighted item is closed
	Highlight first child if the currently highlighted item is open

19.2.3. Edit Control for Multi-valued Properties

The control is for example used to edit the mail addresses of a user and to edit role owners.

19.2.3.1. If the focus is on a value:

CR	Change to editable mode

19.2.4. List

19.2.4.1. If the focus is on a list header cell:

CR	Sort table by that column (for sortable columns only)
	Resort in reverse order if table is already sorted by that column
DOWN	Move the focus to the first row

19.2.4.2. If the focus is on a list body cell:

CR	Change to editable mode (if applicable)
UP	Move the focus to the previous row
DOWN	Move the focus to the next row
PAGE UP	Display the previous page and move the focus one page up if current page is not the first one
	Move the focus to the first entry if current page is the first one
PAGE DOWN	Display the next page and move the focus one page down if current page is not the last one
	Move the focus to the last entry if current page is the last one
POS1	Display the first page and move the focus to the first entry

CR	Change to editable mode (if applicable)
END	Display the last page and move the focus to the last entry
CONTEXT MENU	Opens the context menu for the row having the focus

19.2.5. List Context Menu

ESC	Close the context menu without selecting an item

19.2.5.1. If the focus is on a menu item:

CR	Select the item and close the context menu
UP	Move the focus to the previous item
DOWN	Move the focus to the next item

The keys UP and DOWN work in a round-robin fashion.

20. Search Panel Configuration

This chapter provides details on some selected configuration options of the search panel. For other options refer to the Web Center Reference Guide.

20.1. Filter Attributes

The list of filter attributes for a search panel is configured as part of the tiles definition of the corresponding entry list.

The tiles property with name “attributes” contains the list of filter attributes, separated by semicolons.

Each attribute includes up to four items, separated by colons:

- The LDAP attribute name.
- The message key for the attribute label; the default key is “ldap.attribute.<attribute name>”.
- The attribute type. The type determines the filter operands to be applied to this attribute, like “equals” or “beginsWith”, and defines the way in which the filter values can be entered or selected. The supported types include:
  - General types
    - **date** – for date attributes without time like a user’s start date, with day regarded as GMT day.
    - **day** – like date, but with day extending from GMT start of day -12 hours to GMT start of day + 12 hours.
    - **time** – for date attributes with time, like the time until which a user account is locked out.
    - **number** – for attributes with numeric values.
    - **noSubs** – for attributes which do not (or should not) support filtering by substrings.
    - **sublink** – for filtering by an attribute of a link, while restricting the set of all links to a specific subset. For details, see the section “Filtering by Link Attributes”.
  - Specific types for attributes with fixed value sets, like:
    - **bool** – for Boolean attributes (yes/no)
    - **accountType** – for account attribute dxrType (personal/privileged)
    - **contextType** – for context attribute dxrType (category/general)
    - **projectType** – for project attribute dxrType (like Consulting)
    - **dxrState** – for account or group attribute dxrState (like ENABLED)
    - **dxrTsState** – for account or group attribute dxrTsState (like ENABLED)

    For details, see the section “Filtering by Attributes with a Fixed Value Set”.

    - Types for attributes with proposal lists, like a user's risk level. For details, see the section "Filtering with Proposal Lists".
- The visibility flag: **true** for visible, **false** for hidden. This can be an expression evaluated at run time. Filter attributes are visible by default.

20.1.1. Filtering by Attributes with a Fixed Value Set

The Web Center search panel supports searching for attributes with a fixed value set, like Boolean attributes or attributes whose values are provided by a static proposal list.

Web Center provides some predefined sets like "bool" for boolean attributes represented as "Yes" and "No", or "accountType" with values "Personal" and "Privileged".

For a new value set, first choose an identifier for the list, then assign localized values for each supported language, and finally assign the list to one or more search filter attributes.

Note that localized proposal lists provide an easier and better way to achieve the same. (See the next chapter for details).

20.1.1.1. Defining the Localized Values

The set of localized values is defined as a sub-object of object "messages.texts" in Javascript file **resources/build/messages/<language>/messages.js**. The sub-object's name is the value list identifier.

Each sub-object maps the values to their localized representations. The localized values must be enclosed in single or double quotes. You can spare the enclosing quotes around the value list name and the values themselves as long as they don't conflict with reserved Javascript identifiers or other Javascript code.

Sample:

```
messages.texts = {
    accountType: {
        Personal : "Personal",
        Privileged: "Privileged"
    },
    bool: {
        'TRUE'  : "Yes",
        'FALSE' : "No"
    },
    …
}
```

20.1.1.2. Configuring the Search Filter Attribute

The filter attribute is configured in the search form's Tiles definition by its name, followed by two colons and the value list identifier.

Sample:

```
<definition name=".accountsForm"
            path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action" value="/getAccounts.do"/>
    <put name="attributes"
         value="cn;dxrIsPrimary::bool;dxrType::accountType;…"/>
```

20.1.2. Filtering with Proposal Lists

The Web Center search panel supports searching for attributes whose values are provided by a localized or non-localized proposal list. The proposal list can be static or dynamic, of type String or DN. The list should be of a reasonable size in order not to cause performance issues.

20.1.2.1. Configuring the Search Filter Attribute

The filter attribute is configured in the search form's Tiles definition by its name, followed by two colons and the proposal list identifier. The identifier is the concatenation of object type (from file **objects-config.xml**), property name and the string "proposals", separated by dots, for example "dxrSalutation::dxrUser.dxrSalutation.proposals".

Sample:

```
<definition name=".usersForm"
            path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action" value="/getUsers.do"/>
    <put name="attributes"
         value="cn;dxrSalutation::user.dxrSalutation.proposals;…"/>
```

20.1.3. Filtering by Link Attributes

The Web Center search panel supports searching for attributes that reference other objects, like user manager or role owner.

Let's say you want to search for all groups whose owner has a surname starting with "ab". Then Web Center first searches for possible owners, that is for users whose surname starts with "ab". Then it searches for all groups whose owner matches one of the users found.

Be aware that the first search may hit a size or time limit. The returned list of possible owners is then probably incomplete. The final result will then contain only groups with

owners in the incomplete list.

Sometimes one wishes to restrict the set of referenced objects to a specific subset. For example, users might want to filter their task list by the name of the privilege that is the task resource but only if the resource is a role. This can be done by assigning the type sublink to the search attribute.

20.1.3.1. Configuring the Search Filter Attribute

The filter attribute is configured in the search form's Tiles definition by its name, followed by an at sign and the name of the attribute of the linked objects, for example "owner@sn".

Searching by the owner's surname:

```
<definition name=".groupsForm"
    path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action"     value="/getGroups.do"/>
    <put name="attributes" value="cn;description;owner@sn;…"/>
…/>
```

Searching by the manager's surname or given name:

```
<definition name=".usersForm"
    path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action"     value="/getUsers.do"/>
    <put name="attributes" value="cn;sn,gn;manager@sn,manager@gn;…"
…/>
```

Searching by the owner's or role administrator's surname:

```
<definition name="rolesForm"
    path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action"     value="/getRoles.do"/>
    <put name="attributes"
         value="cn;owner@sn,dxrRoleAdmin@sn:ownerOrAdmin;…"
…/>
```



You must configure attribute **dxrRoleAdmin** in file **WEB-INF/config/filters-config.xml**, and you must assign a label for the search filter attribute to key **ownerOrAdmin** in message file **text.properties**.

Filtering the task list by the name of roles that are task resources:

```
<definition name=".worklistForm"
    path="/WEB-INF/jsp/view/tiles/objectList.jsp">
    <put name="action"     value="/getUsers.do"/>
    <put name="attributes"
         value="dxrResourceLink.role@cn:
                requestwf.workflow.resource.role:sublink;…"
…/>
```

20.1.3.2. Configuring the Search for the Linked Objects

You can define some individual and some general options for the search for the linked objects. The details are described in chapter “User Interface Configuration / Search Filter Configuration” of the Web Center Reference Guide.

21. Adding Languages

This chapter describes how to add a language to a Web Center application. It shows which files must be translated and gives some guidelines that must be followed during translation.

The chapter also describes how to add the new language to the language chooser and how to define the new language as the default language.

This chapter does not cover localizable items that are just used by Web Center but are localized in other components like the DirX Identity database; for example, descriptions and attribute names of request workflow tasks.

21.1. Overview

21.1.1. Files to Be Translated

The list of files to be translated includes:

- An HTML file displayed when the browser's Javascript support has been disabled.
- A Javascript message file **messages.js**
- A Java properties file **text.properties**
- HTML snippets with extension **html**
- JSP snippets with extension **jsp**
- Pages displayed in cases of severe errors.

The following figure shows the files and their locations within a Web Center application:

```
resources
    build
        html
            noscript.html
        messages
            <language>
                 messages.js

WEB-INF
    classes
        resources
            languages
                <language>
                    text.properties
                    *.html
```

```
                    *.jsp
                    <any subfolder, recursively>
                        *.html
                        *.jsp

WEB-INF
    jsp
        view
            error
                errorMessages.txt
                forbidden.html
                severeErrorPage.html
```



Often, you don't have to translate all the files listed here. You can especially spare those subfolders of **WEB-INF/classes/resources/languages/<language>** that relate to types of objects you don't support in your application.

If, for example, you don't use business objects, you can ignore the files in folder **WEB-INF/classes/resources/languages/<language>/bo**.

21.1.2. How to Proceed

To prepare and execute translation:

- Identify the ISO 639 alpha-2 or alpha-3 code of your new language.You can find the codes on the Internet.Examples are **it** for Italian and **nl** for Dutch.
- Copy the two language folders for an existing language to new folders for your new language.For example, copy **resources/build/messages/en** to **resources/build/messages/nl** and **WEB-INF/classes/resources/languages/en** to **WEB-INF/classes/resources/languages/nl**.
- Save a backup of the files that must be directly modified.These files are listed in the previous step and are located in the folder **resources/build/html** and in folder **WEB-INF/jsp/view/error**.
- Translate each message file as described in the next section.
- Configure the language chooser and, if applicable, set the default language to your new language.

21.2. Translating Files

21.2.1. Message Files

21.2.1.1. File messages.js

The Javascript file **messages.js** contains message texts evaluated directly by the browser.

The first definition assigns the actual language code to the attribute language of the message object.Set the value to the code of your new language, like

```
messages.languages = "nl";
```

for Dutch.

The main section “messages.texts” contains a hierarchical number of message sections represented in JSON format. Each object is enclosed in curly braces and contains a list of attribute value pairs. A value is either of a basic Javascript type like String, Number or Boolean, or again a JSON object. Attribute and value are separated by a colon. If the attribute contains spaces or matches a Javascript reserved name like true or false, it must be enclosed in single or double quotes. Otherwise, the quotes are optional. Separate successive attribute value pairs within a section by a comma, but take care not to add a comma after the last attribute value pair within a section.

The section “messages.custom.texts” is used to define custom alternatives for the texts in “messages.texts”. The texts are usually not customized here but in separate custom files.

Don’t change the attributes since they are used to access the message texts in other Javascript files. Just translate the values into your new language.

Each double quote with a value must be escaped with a backslash. Non-ASCII characters in the values must be replaced with their Unicode representations.

```
nonASCII: "A value with umlaut: \u00F6.",
quoted: "A \"quoted\" value",
```

The file is located in the folder *resources/build/messages/**language*.

Use the translation utilities delivered with Web Center to translate the message file without having to manually replace non-ASCII characters with their Unicode representations. The utilities are described below.

21.2.1.2. File text.properties

The Java properties file **text.properties** contains message texts evaluated on the server side.

Each message consists of a key and a text, separated by an equality character. The keys are fixed and mustn’t be changed. Non-ISO-8859-1 characters in the texts must be replaced with their Unicode representations.

The file is located in folder **WEB-INF/classes/resources/languages/<language>**.

Use the translation utilities delivered with Web Center to translate the properties file without having to manually replace non-ISO-8859-1 characters with their Unicode representations. The utilities are described below.

21.2.2. Snippets

Each language directory contains a set of HTML snippets (extension **.html)** and JSP snippets (extension **.jsp)**. Each snippet defines the introductory text that is displayed at the top of the application-specific section of a Web Center page. The text briefly describes the data that is displayed on the page, or the functionality provided by the page. There's usually a separate snippet per object type (like user, role and password policy) and per function (like list, summary and modify).

By default, the snippets are read from the local file system using the default file encoding of the Java version. This will usually not work correctly for snippets containing non-ASCII characters. That's why the English and German snippets use HTML references for non-ASCII characters, like "ä" for "ä". This is of course inconvenient especially for languages with character sets quite different from ASCII.

21.2.2.1. JSP Snippets

JSP snippets contain HTML code and JSP code that inserts some dynamic content into the page, like the value of a configuration parameter.

When translating a JSP snippet into another language, create and store the snippet in UTF-8 encoding, and add a metatag-instruction at the beginning of the file specifying the encoding used. Then you can use any character that can be UTF-8 encoded, as shown in the following sample:

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<h2 class="headlineText">
    Welcome to DirX Identity Web Center
    ÄÖÜ - αβγ - йкл - フヨㅂ
</h2>
```

Don't change any JSP code in the snippets.

21.2.2.2. HTML Snippets

HTML snippets contain pure HTML code. The snippet encoding is not defined in the HTML snippets itself but per language in file **webCenter.properties**:

```
resources.html.fileEncoding.de = ISO-8859-1
resources.html.fileEncoding.en = ISO-8859-1
```

```
resources.html.fileEncoding.ja = UTF-8
```

When translating HTML snippets into another language, add a configuration parameter for that language with an appropriate encoding. Then create and store the snippets in the specified encoding.

21.2.3. Error Pages

The error pages are displayed in cases of severe errors where even the end user's language sometimes cannot be detected. Therefore, there isn't one error page per language. Instead, each page displays a message in all the supported languages.

21.2.3.1. File noscript.html

This file is displayed when the browser's Javascript support is disabled.

```
<h2 class="headlineText" style="color:red;margin-top:50px">
    The requested page cannot be displayed since Javascript is
disabled in your browser!
</h2>
<h2 class="headlineText" style="color:red;margin-top:25px">
    Die angeforderte Seite kann nicht angezeigt werden, da Javascript
in Ihrem Browser deaktiviert ist!
</h2>
```

Just add a new HTML <h2 class="headlineText"> element with a corresponding text in the new language to the HTML body.

The message text is HTML code. Replace any special and any non-ASCII character with its corresponding HTML reference.

The file is located in folder **resources/build/html**.

21.2.3.2. File errorMessages.txt

This file is used when processing a request fails for a severe reason. The error page tries to display a message in the end user's preferred language.

```
<%
    messages.put("de","Ihre Anfrage konnte nicht bearbeitet
werden.");
    messages.put("en","Your request could not be processed.");

    defaultLanguage = "en";
```

```
%>
```

The file contains some Java code that fills a message map with name “messages”. Just put a new message with the code of the new language as key and the corresponding message text as value into the map.

The message text is a Java string. Escape each double quote within the text with a backslash, and replace any non-ASCII character with its Java Unicode character representation, like in

```
messages.put("en",
    "A \"quoted\" message with an umlaut \u00e4.");
```

The file is located in folder **WEB-INF/jsp/view/error**.

21.2.3.3. File forbidden.html

This file is displayed if access to a Web Center page has been rejected for security reasons.

```
<body>
    <h1>Access to the requested resource has been forbidden.</h1>
    <h1>Der Zugriff auf die angeforderte Seite wurde abgelehnt.</h1>
</body>
```

Just add a new HTML <h1> element with a corresponding text in the new language to the HTML body.

The message text is HTML code. Replace any special and any non-ASCII character with its corresponding HTML reference.

The file is located in folder **WEB-INF/jsp/view/error**.

21.2.3.4. File severeErrorPage.html

This file is displayed as a last resort when processing a request fails such that a user-friendlier and less disruptive page cannot be displayed.

```
<td colspan="3" class="leftContent" style="…">
    <h2 class="headlineText">Your request could not be
processed.</h2>
    <h2 class="headlineText">Ihre Anfrage konnte nicht bearbeitet
werden.</h2>
</td>
```

Just add a new HTML <h2 class="headlineText"> element with a corresponding text in the new language to the HTML body.

The message text is HTML code. Replace any special and any non-ASCII character with its corresponding HTML reference.

The file is located in the folder **WEB-INF/jsp/view/error**.

21.2.4. Character Representations

21.2.4.1. HTML Code

In HTML code, you can use any ASCII character. Some characters, however, have a special meaning in HTML and should be replaced with their corresponding HTML entity references to avoid misinterpretations.

Table 1. HTML Entity References for Special HTML Characters

Character	HTML Entity Reference
Double quote: “	"
Ampersand: &	&
Less than: <	<
Greater than: >	>

You can also use non-ASCII Unicode characters like umlauts, Greek and Cyrillic characters, and characters from Asian languages. These characters must also be replaced with their corresponding HTML entity or numeric references.

Table 2. HTML References for Sample non-ASCII Characters

Character	HTML Entity Reference	Numeric Reference
ä	ä	ä
ö	ö	ö
ü	ü	ü
Ä	Ä	Ä
Ö	Ö	Ö
Ü	Ü	Ü
ß	ß	ß

For a complete list of HTML numeric references, see the file *install_path*/**web/instances/webCenter-***domain*/**tools/TranslationUtilites/files/HTMLNumericReferences.html.**

21.2.4.1.1. Samples

```
<h1>This is a &quot;quoted&quot; message.</h1>
<h1>This is a text with the greek letter gamma &#x393; and an umlaut
&auml;.</h1>
```

21.2.4.2. Java and JSON Unicode Character Representations

Replace any non-ASCII character within a Java String or JSON value with its Unicode escape sequence, which is “\u”, followed by the 4-digit hexadecimal character value. Alphabetic hex digits (“a” - “f”) can be either lower case or upper case.

Table 3. Unicode Character Representation for Sample non-ASCII Characters

Character	Unicode Representation
ä	\u00e4
ß	\u00df
Γ	\u0393
Δ	\u0394
Б	\u0411
Д	\u0414
▮	\u0e01
▮	\u0e02

For a complete list of Unicode character representations, see file *install_path*/web/instances/webCenter-*domain*/tools/TranslationUtilites/files/JavaUnicodeCharacterRepresentations.html.

21.2.4.2.1. Sample

"This is a string with the greek letter gamma \u0393; and an umlaut \u00e4."

21.2.5. Translation Utilities

When translating a message file into a new language, you probably don’t want to manually replace all non-ASCII or non-ISO-8859-1 characters with their substitutes. It’s much more convenient to enter the message texts as usual, for example in an editor that supports UTF-8, and then automatically generate a message file with correct substitutions. Web Center delivers some Windows batch scripts that convert message files from ASCII or ISO-8859-1 to UTF-8 (or any other encoding), and vice versa. The scripts are based on the JDK utility **native2ascii.exe**, so you need a JDK (not just a JRE).

The scripts are located in the folder *install_path*/web/instances/webCenter-*domain*/tools/TranslationUtilites/bin. Use the scripts in subfolder **text.properties** to

translate **text.properties** files and the ones in subfolder **messages.js** to translate **messages.js** files.

First, fix the JDK path and choose your preferred encoding for editing in file **setEnv.bat**. The predefined encoding is “utf-8”, which is supported, for example, by Notepad and jEdit.

21.2.5.1. text.properties

To translate a **text.properties** file, proceed as follows:

- Change to subfolder **text.properties**.
- Copy your original **text.properties** file into the folder (overwriting the delivered sample file).
- Run script **createEditableFile.bat**. The script creates file **text-editable-encoding.properties**. In the file name, encoding is replaced with the selected encoding, like “utf-8”.
- Open the created file in a suitable editor and then translate the message texts. When finished, save the file.
- Run script **createWebCenterFile.bat**. The script creates file **text-WebCenter.properties**. Check if the special characters have been correctly replaced.
- Copy the file **text-WebCenter.properties** to its destination folder within your Web Center application, renaming it back to **text.properties**.

21.2.5.2. messages.js

To translate a **messages.js** file, proceed as follows:

- Change to the subfolder **messages.js**.
- Copy your original **messages.js** file into the folder (overwriting the delivered sample file).
- Run the script **createEditableFile.bat**.The script creates file **messages-editable-encoding.js**.In the file name, encoding is replaced with the selected encoding, like “utf-8”.
- Open the created file in a suitable editor and then translate the message texts.When finished, save the file.
- Run the script **createWebCenterFile.bat**.The script creates file **messages-WebCenter.js**.Check if the special characters have been correctly replaced.
- Copy the file **messages-WebCenter.js** to its destination folder within your Web Center application, renaming it back to **messages.js**.

21.3. Configuration Tasks

21.3.1. Extending the Language Chooser

To add a new language to the language chooser that is displayed in the header section of each Web Center page, add the language code and its display label to the message with

key "application.option.language" in the message file **text.properties**. Do this for each supported language.

```
application.option.languages = en:English;de:Deutsch;nl:Dutch
```

21.3.2. Changing the Default Language

The default language is defined in file **WEB-INF/web.xml** by configuration parameter **javax.servlet.jsp.jstl.fmt.fallbackLocale**. To change the default language, just assign the new language code.

```
javax.servlet.jsp.jstl.fmt.fallbackLocale = nl
```

21.3.3. How a User's Language is Determined

When a user starts a Web Center session, Web Center applies the following rules with descending priority in order to determine the user's language:

- The request includes a Web Center language cookie and the language specified in the cookie is valid.
- The request includes an Accept-Language header and one of the requested languages is valid. The languages in the header are evaluated according to their quality factors. The browsers usually let a user define his preferred languages via a configuration option.
- The default language.

The language is stored in the user's session and holds throughout the session unless the user chooses another language from the language chooser.

Web Center also tries to persistently store the language in a browser cookie (the language cookie) for subsequent sessions. But since browser cookies are not reliable, the language chosen for one session may be lost until the next session.

22. Confirmations

22.1. Overview

If a user submits a form it is sometimes desirable to request an additional confirmation because by submitting the form the user might inadvertently trigger an action with severe and unforeseen consequences.

Web Center supports such confirmations. For each form property, you can define conditions which are checked if the form is submitted. If a condition is met, a message is displayed and a confirmation is requested. The user can then either confirm or cancel the request.

The configuration of a confirmation consists of the name of a Javascript function and the key of the confirmation message. The task of the Javascript function is to check the condition for displaying the message.

22.2. Condition Check Functions

A confirmation is requested if a certain condition is met. The condition is checked by a Javascript function. The function is called whenever the user submits the corresponding form. It is also called once immediately after form creation. The function can then for example store the initial value of a form field for later use in subsequent calls when the form is submitted.

Each function must be registered with a unique identifier. The identifier is used to specify the check function when configuring a confirmation for a form property.

22.2.1. Check Function Interface

A function for checking a condition must implement the interface

```
checkCondition (init, id, form, state)
```

Function name and argument names are of course arbitrary.

The arguments are

- **init –** Whether the function is called during form creation (true) or when the end user submits the form (false)
- **id –** The HTML id of the form field
- **form –** The HTML form element
- **state** – The state returned from the initial call.

The initial call may return any value, like a boolean, a string or an object. The value is passed transparently as argument **state** to subsequent calls.

Each subsequent call must return true if the condition is met, and false if not.

22.2.2. Predefined Checks

Web Center is shipped with two predefined condition checks. The first one checks if the value of a single-valued text field has been deleted or not; it is implemented by the function “confirmationConditions.valueDeleted“:

```
valueDeleted: function(init,id,form,wasEmpty) {
    var f = GF(id);
    if (f && (init || !wasEmpty))
        return isEmptyString(f.value);
    return false;
}
```

The initial call returns whether the field is initially empty or not. The result is then passed as 4th argument (“wasEmpty“) to subsequent calls which return true if the current value is empty while the initial one was not, which means if the value was deleted.

The second predefined function checks if role parameters are specified or not; it is implemented by the function “roleParams.isEmpty”.

Note that the check functions are usually deeply intertwined with the renderers used to display the corresponding form fields. Therefore, you cannot simply use them for form fields displayed by other renderers.

22.2.3. Registering Checks

You must register each condition check with a unique identifier by calling “confirmationConditions.register”, passing an array with the identifier and the check function. You can register one or more checks in one call.

The predefined condition checks are registered with identifiers “onDeleteValue” and “onEmptyRoleParams”:

```
confirmationConditions.register(
    [ "onDeleteValue", confirmationConditions.valueDeleted ],
    [ "onEmptyRoleParams", roleParams.isEmpty ]
);
```

Registering a single check goes like this:

```
confirmationConditions.register(
    [ "onEmptyRoleParams", roleParams.isEmpty ]
```

```
);
```

22.3. Form Property Configuration

Configuring a confirmation for a form field is easy. Form properties support the attribute “confirmation”, whose value should be the check identifier followed by the key of the confirmation message to be displayed if the condition is met. Identifier and message key must be separated by a colon.

Sample:

To get a confirmation message whenever a user’s manager is deleted, configure the form property like this:

```
<form-property name="manager" type="java.lang.String"
    fieldRenderer="userSearch"
    …
    confirmation="onDeleteValue:manager.deleted"
/>
```

When defining the message text, use “\n” for line breaks. A backslash at the end of a line in the message file indicates that the message text extends over the next line:

```
manager.deleted = You've deleted the user's manager.\n\n\
                  Are you sure?
```

23. Validation of Editable Input Fields in Tables

23.1. Overview

Web Center supports validation of editable input fields in tables. A validation may involve just a single table cell, or several table cells within the same row. A validator may for example make sure that a date lies within a specified range, or that some dates are in proper order.

23.2. Samples

23.2.1. Date Validations

Let's say we have a table with columns for editable start dates and end dates. We want to make sure that

- Dates are entered in the correct format.
- Newly entered or changed dates don't lie in the past.
- The start date always precedes the end date.

To achieve this we add three validators to the table's **checks** attribute and assign some identifying columns to the table's **namingColumns** attribute.The attributes are described in greater detail in the following chapters.

```
<form-property name="assignedGroups" ...
   type="com.siemens.webMgr.model.DirectoryEntryBean[]"
   checks="date:assign.dxrStartDate:['TODAY']:lowerBoundedDate;
           date:assign.dxrEndDate:['TODAY']:lowerBoundedDate;

dates:assign.dxrStartDate,assign.dxrEndDate:[true]:beginEndDateLess"
   namingColumns="$displayName,targetSystem.cn">
    <data-property name="$displayName" type="java.lang.String"/>
    <data-property name="description" type="java.lang.String"/>
    <data-property name="targetsystem.cn" type="java.lang.String"/>
    <data-property name="assign.dxrStartDate" type="java.util.Date"
           label="ldap.attribute.dxrStartDate" width="8%"
                  readonly="false" sortable="false"/>
    <data-property name="assign.dxrEndDate" type="java.util.Date"
           label="ldap.attribute.dxrEndDate" width="8%"
                  readonly="false" sortable="false"/>
```

```
</form-property>
```

23.2.2. Time Validations

Now let's consider a table with columns for editable start times and end times. We want to make sure that

- Times are entered in the correct format.
- Newly entered or changed start times don't lie in the past.
- The start time always truly precedes the end time.
- The end time is no later than the last day of 2020.

To achieve this we add three validators to the table's **checks** attribute. The attribute is described in greater detail in the following chapters.

```
<form-property name="users" ...
   type="com.siemens.webMgr.model.DirectoryEntryBean[]"
   …
   checks="time:startTime:['NOW']:lowerBoundedTime;
           time:endTime:['','2020-12-31T23\:59Z']:upperBoundedTime;
           times:startTime,endTime:[true]:customStartEndTime">
    <data-property name="$displayName" type="java.lang.String"/>
    <data-property name="startTime" type="java.util.Date"
            label="ldap.attribute.startTime"
                  readonly="false" cellRenderer="time"/>
    <data-property name="endTime" type="java.util.Date"
            label="ldap.attribute.endTime"
                  readonly="false" cellRenderer="time"/>
</form-property>
```

23.3. Configuration

23.3.1. Checks

The **checks** attribute accepts the semicolon-separated list of validators to be applied to the table.

Each validator consists of

- The name under which the validator is registered.
- The data property names of the table columns the validator applies to.
- The validator arguments (a Javascript array, which may be empty).

- The key for the tooltip text (optional).

The validator components are separated by colons.

23.3.1.1. Sample

In the above sample, the first validator checks the format of start dates and rejects start dates in the past:

```
date:assign.dxrStartDate:['TODAY']:lowerBoundedDate;
```

The second validator performs the same checks for end dates:

```
date:assign.dxrEndDate:['TODAY']:lowerBoundedDate;
```

The third validator makes sure that a start date precedes the corresponding end date:

```
dates:assign.dxrStartDate,assign.dxrEndDate:[true]:beginEndDateLess
```

23.3.2. Naming Columns

If a user tries to submit a form with an invalid value in a table cell, Web Center displays a corresponding error message. In order to make clear which entry (row) in the table the message refers to, the message includes the values of one or more table cells that identify the entry.

The attribute **namingColumns** contains the names of the data properties that identify an entry, separated by commas.

By default, only the value in the first column is displayed.

23.3.2.1. Sample

In the above sample, the groups are identified by their name and their target system, so these values should be displayed in error messages:

```
namingColumns="$displayName,targetSystem.cn"
```

23.3.3. Validators

23.3.3.1. Single-Property Validators

A single-property validator consists of

- A validator name; the name must be a valid Javascript variable name; required.
- A list of validator arguments; optional.
- A Javascript object with member:
  - **check** – A function that checks the validity of a value. The function accepts the value as its first argument, and the validator arguments as additional arguments. The function must return “false” if the check fails, and “true” otherwise.
- Tooltip texts; per supported language; optional. Tooltips may contain placeholders to be replaced with the actual validator arguments at runtime. “${0}” is replaced with the first argument, “${1}” with the second, etc. The placeholders for date arguments are “${0d}”, “${1d}” etc.

Notes:

- A custom tooltip must be assigned to

  ```
  messages.custom.texts.tooltip.format.<validator name>
  ```

  You can define a separate tooltip for Firefox. This is sometimes useful for tooltips including new lines, since Firefox and Internet Explorer render new lines in a different way (Firefox just ignores new lines):

  ```
  messages.custom.texts.tooltip.format.ff.<validator name>
  ```

- You can use single-property validators for form input fields to validate editable table cells, like for example **date**. Note that the regular expression is ignored. Only the check function is called.
- Register custom validators via function **validators.register**.

23.3.3.2. Mutliple-Property Validators

A multiple-property validator consists of

- A validator name; the name must be a valid Javascript variable name; required.
- A list of validator arguments; optional.
- A Javascript object with member:
  - **check** – A function that checks the validity of the property values. The function accepts the following arguments:
    - The nodes to be checked (a Javascript array). Each node is an object with the following fields:
      - **id -** The HTML id of the table cell.
      - **value –** The value to be checked.
    - The second argument is always true since table validators are not triggered by

key up events.

- The optional validator arguments.

  The function must return an object with two fields named “ok“ and “inv“. “inv“ is an array containing the indexes of the nodes to be added to the list of invalid properties, while “ok“ contains the indexes of the nodes to be removed from that list. Any property not referenced in the arrays is left untouched.

  In addition to that, the function may also return tooltip keys for invalid cells. The keys have to be passed in field “msg“ of the returned object; The n^{th} item in “msg“ corresponds to the n^{th} node. The tooltips override the tooltips for the validator defined below.

- Tooltip texts; per supported language; optional. You can define a single tooltip for all properties, or different tooltips per property. Tooltips may contain placeholders to be replaced with the actual validator arguments at runtime. ${0} is replaced with the first argument, ${1} with the second, etc. The placeholders for date arguments are ${0d}, ${1d} etc.



- Each custom tooltip must be assigned to

  ```
  messages.custom.texts.tooltip.format.<tooltip name>
  ```

  You can define separate tooltips for Firefox. This is sometimes useful for tooltips including new lines, since Firefox and Internet Explorer render new lines in a different way (Firefox just ignores new lines):

  ```
  messages.custom.texts.tooltip.format.ff.<tooltip name>
  ```

- You can use multiple-property validators for form input fields to validate editable table cells, like for example **dates**. In complex cases, however, it might be more appropriate to write a specific validator; for an example, see validator **certDates**.
- Register custom validators via function **validators.register**.

24. Customizing the Help Link in the Menu Bar

24.1. Overview

The Web Center menu bar contains a help button that displays a help file, for example a PDF, in a separate browser window. Since loading the PDF directly into the help window shows some minor inconveniences, Web Center first loads an HTML wrapper into the help window which then in turn loads the help file.

The wrapper allows for setting a localized window title and a shortcut icon. Also, without the wrapper it's not possible for the main Web Center page to bring the help window to the front if the user clicks the help button and the help window is already open but hidden behind other windows (in Internet Explorer).

24.2. Customizing the Help Button in the Menu Definition

In file **WEB-INF/config/identity/menu-defs.xml**, you can define and customize the help button.

24.2.1. Defining the Help Menu

The menu is defined as usual. The only specific part is the value of attribute "titleItem" which specifies

- The help file location. This can be
  - The path (within Web Center) of the HTML wrapper.
  - The path (within Web Center) of the help file (no wrapper).
  - The URL of an external help file.

  The location may contain the expression "${language}" which is replaced with the user's Web Center language at runtime.
- **targetName** - The name of the help window. If you define the same window name for more than one Web Center application, the applications' help files will all be loaded into the same window. If you define a different name per application, each application loads its help file in a separate window.
- **targetType** - The identifier of the configuration property for the window properties and shortcut key (see section "Customizing the Help Window" below). The window properties are evaluated only when the window is opened, not when loading a help file into an already open help window. Therefore, applications sharing the same help window should also have the same window properties.

The default definition for the help button is:

```
<!-- Help menu -->
<definition name=".menuHelp" extends=".menu">
    <put name="msgPrefix" value="help"/>
    <put name="titleItem"
         value="help:/resources/help/${language}/WebCenterHelp.html::
                targetName=webCenterHelp,targetType=help"/>
</definition>
```

24.2.2. Adding the Help Menu to a Menu Bar

The help button is by default placed before the login button in the main menu bar:

```
<definition name=".defaultMenubar">
    <put name="items" value=".menuHome;sep;…;
                             .menuTools;sep;
                             .menuHelp;.menuLogout;nl;
                             …/>
</definition>
```

You can move it to another position within the main menu bar, or remove it from the main menu bar and add it to another one. If you don't need the help button at all, just remove it here.

24.3. Providing the Help Files

Copy your help files to folder **resources** or one of its sub folders. The empty default help files are **resources/help/en/WebCenterHelp.pdf** and **resources/help/de/WebCenterHelp.pdf**.

24.4. Customizing the Help Window

24.4.1. General Properties

In file **resources/build/config/config.js**, you can configure some properties of the help window and the shortcut key for help. The properties are transparently passed to the Javascript function window.open, so confer to any documentation of that function for a complete list of supported properties and valid property values.

By default, the key F1 opens the Web Center help window. The window is 800 pixels high, 1000 pixels wide, and its top left corner is 50 pixels left and 100 pixels below the top left corner of the screen. The window is resizable but shows no scroll bars.

```
target: {
```

```
    help: {
        key:   "F1",
        props:

"height=800,width=1000,left=50,top=100,scrollbars=no,resizable=yes"
    }
}
```



F1 usually opens the browser's help page. When assigned to the Web Center help F1 does no longer trigger the browser help.

24.4.2. Per-Language Properties

The provided HTML wrappers allow for setting the window title and the shortcut icon. They also contain the help file name. The wrappers for the standard Web Center applications are located in folders **resources/help/de** and **resources/help/en**.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
 <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE9" />
 <title>Web Center Help</title>
 <link rel="shortcut icon" href=
"../../images/logos/DirXIdentity.ico"/>
</head>
<body style="padding:0;margin:0;overflow:visible">
  <iframe src="WebCenterHelp.pdf"
          frameborder="0" scrolling="0"
style="width:100%;height:100%">
  </iframe>
</body>
</html>
```

Adjust the title, the "href" attribute of the shortcut link and the "src" attribute of the iframe according to your needs.

25. Rendering Attribute Values via Icons

25.1. Overview

Web Center supports rendering read-only attributes with a fixed set of values via icons. Each icon represents an attribute value.

We demonstrate how to use the renderer by means of a sample.

25.2. Attribute Risk Level

The user attribute risk level is set by the risk calculation algorithm to one of the values 0, 1, 2 and 3.

For display purposes, the numeric values are mapped to localized display values by localized proposal lists in folder **cn=Risk Levels, cn=Nationalization, cn=Proposal Lists, cn=Customer Extensions, cn=Configuration, cn=*domain***.

The English display values are for example “No risk” (0), “Low” (1), “Medium” (2) and “High” (3).

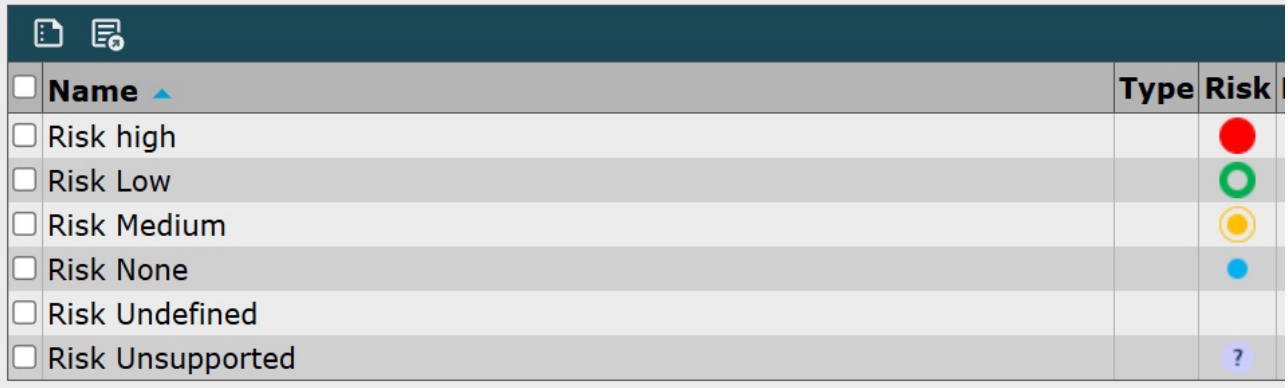
Risk level checks can be activated or deactivated via the domain configuration flag “Risk check active”.

The LDAP attribute name for risk levels is dxrRskLevel (not dxrRiskLevel !)

25.3. Displaying the Risk Level in Web Center

Web Center displays the risk level on user summary pages, in user lists and many other pages. Assignment approval pages indicate that a user’s risk level will change to “High” should the assignment be approved.

Here’s a user list showing the default risk level icons:



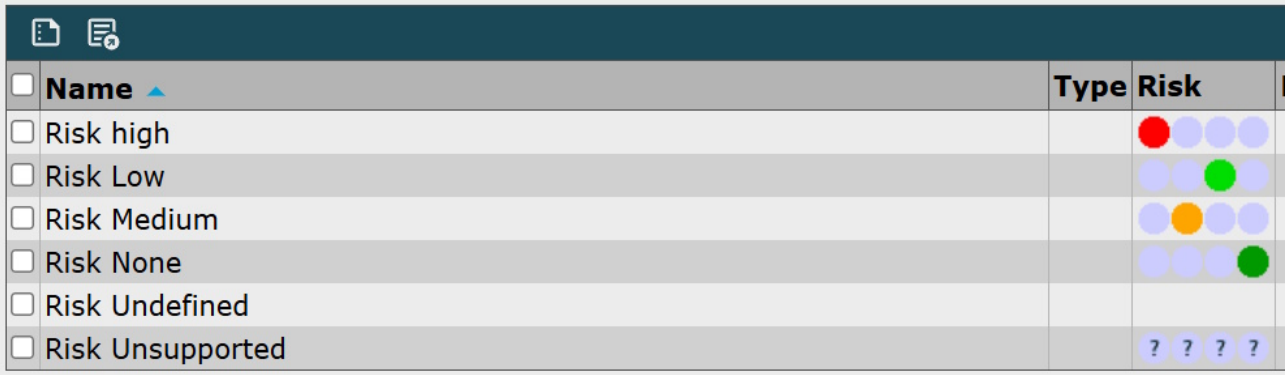
You can see two additional risk levels:

- - indicates risk levels with no value at all (which means the risk level has not been

calculated yet)

- ? - indicates risk levels outside the supported value set (0, 1, 2, and 3).

An alternative set of icons mimics traffic lights:



Web Center displays risk levels only if risk level checks are enabled. For that reason, it makes the value of the domain configuration flag “Risk check active” available in the application-scoped Boolean attribute “com.siemens.webMgr.riskCheckEnabled”.

25.4. Configuring Web Center

25.4.1. Defining the Icon Renderers

The renderers are defined in file

WEB-INF/config/renderers-config.xml.

25.4.1.1. The Base Icon Renderer:

The renderer is based upon an HTML snippet for displaying an icon as HTML form field, and a Javascript snippet for diplaying an icon in a list column.

```
<renderer id="icon"
    className=
        "com.siemens.webMgr.taglib.view.renderers.IconRenderer"
    defURL="/WEB-INF/snippets/icon/icon.htm"
    jsURL="/WEB-INF/snippets/icon/icon.js">
    <renderer-property
        name="titleUndefined" value="icon.title.undefined"/>
    <renderer-property
        name="titleUnsupported" value="icon.title.unsupported"/>
    <renderer-property
        name="iconStyleClass" value="iconProperty"/>
</renderer>
```

The renderer specific properties are

- **folder –** the name of the folder containing the icons. Specify the name relative to the application's document root folder.
- **iconPrefix –** the icon name prefix.
- **iconSuffix –** the icon name suffix.
- **iconUnsupported –** the name of the icon representing unsupported property values.
- **iconUndefined –** the name of the icon representing properties with no value.
- **titleUnsupported –** the tooltip for the icon representing unsupported property values.
- **titleUndefined –** the tooltip for the icon representing properties with no value.
- **iconStyleClass –** the CSS class name for the icons.

The base renderer defines just default tooltips and the CSS style sheet for the icons, but leaves all other properties unspecified.

The list renderer supports two special properties that control how to react on clicking an icon:

- **link:true –** open the entry in the corresponding row
- **toggleSelection:true –** toggle the selection state of the entry in the corresponding row

By default, a click will have no effect at all. Since the way how to react depends on the specific use case, the properties are usually set via an attribute of the risk level's data property in the list configuration (see chapter "Configuring Forms" below for an example).

25.4.1.2. The Risk Level Renderer:

Renderer "riskLevel" extends the base icon renderer:

```
<renderer id="riskLevel" extends="icon">
    <renderer-property
        name="folder" value="resources/images/riskLevel/discs"/>
    <renderer-property name="iconPrefix"      value="level-"/>
    <renderer-property name="iconSuffix"      value=".png"/>
    <renderer-property name="iconUnsupported" value=
"unsupported.png"/>
    <renderer-property name="iconUndefined"   value="undefined.png"/>
    <renderer-property
        name="titleUndefined" value="icon.title.notCalculated"/>
</renderer>
```

The icons representing the risk level values are expected to be in folder **resources/images/riskLevel/discs** with names

- **level-0.png**
- **level-1.png**
- **level-2.png**
- **level-3.png**
- **unsupported.png**
- **undefined.png**

To activate the alternative traffic light icons, just change the value of renderer property “folder”:

```
<renderer-property
    name="folder" value="resources/images/riskLevel/trafficLights"/>
```

25.4.2. Setting the Default Renderer

The default renderers are defined in file

WEB-INF/config/defaultRenderer.properties.

Renderer “riskLevel” is assigned as default renderer to property “dxrRskLevel”:

```
dxrrsklevel = riskLevel
```

25.4.3. Adjusting the Icon Layout via CSS

CSS styles are defined per font size in file

resources/build/styles/*fontSize*/styles.css.

The icon layout is controlled by CSS class “iconProperty”. The class just sets the icon height to 18 pixels.

```
.iconProperty {
    height:18px;
}
```

25.4.4. Providing the Icons

Copy all icons with their proper names to the folder configured for the renderer. The icons are expected to be of height 18 pixels; otherwise they might look somehow distorted.

You can change the expected height by modifying CSS class “iconProperty” or by assigning your own CSS class to the icon renderer but you should check whether the result meets

your expectations.

25.4.5. Configuring Forms

Form fields and table columns are configured in configuration files

WEB-INF/config/.../forms-config.xml.

25.4.5.1. Form Property

To display the risk level as an HTML form field, just add a corresponding form property to the form configuration.

```
<form-property
    name="dxrRskLevel"
    type="java.lang.String"
    readonly="true"
    visible=
        "${applicationScope['com.siemens.webMgr.riskCheckEnabled']}"
/>
```

The field is visible only if risk checks are enabled for the domain.

25.4.5.2. Data Property

To display the risk level in a list column, just add a corresponding data property to the list configuration.

```
<data-property
    name="dxrRskLevel"
    type="java.lang.String"
    align="center"
    width="1%"
    rendererProperties="link:true"
    visible=
        "${applicationScope['com.siemens.webMgr.riskCheckEnabled']}"
/>
```

Again, the column is visible only if risk checks are enabled for the domain. Clicking an icon will open the entry in the corresponding row since renderer property “link” is set to “true”.

25.4.6. Localizing Labels and Tooltips

The localized texts are defined per language in message file

WEB-INF/classes/resources/*language*/text.properties.

25.4.6.1. Icon Tooltips

The message keys are referenced in the icon renderer definition.

```
icon.title.undefined     = No value defined
icon.title.notCalculated = Not calculated
icon.title.unsupported   = Unsupported value: {0}
```

The argument {0} in the last message is replaced with the actual risk level value at runtime.

25.4.6.2. Labels

The label for attribute dxrRskLevel is assigned to the default message key for attribute labels:

```
ldap.attribute.dxrRskLevel = Risk
```

26. Enabling User Self-Registration

26.1. Overview

The user self-registration is by default disabled for security reasons. This section describes how to enable it.

26.2. Enable Self-Registration Section on Login Page

To enable the self-registration section on the login page, set the property “loginForm.showRegister” in file **WEB-INF/config/webCenter.properties** to **true**:

```
loginForm.showRegister = true
```

26.3. Activate the Self-Registration Actions

Activate the self-registration actions “/startRegistration” and “/finishRegistration” in file **WEB-INF/config/workflows/struts-config.xml** by removing the enclosing XML comment markers “<!--” and “-->”.

26.4. Set the Self-Registration Password

Add the clear text password of user ANYONE (cn=ANYONE,cn=<domain>) with key “ANYONE” to file **WEB-INF/password.properties**. The password will be encrypted on next restart of Web Center.

```
ANYONE=<cleartext password>
```

DirX Product Suite

The DirX product suite provides the basis for fully integrated identity and access management; it includes the following products, which can be ordered separately.



DirX Identity

DirX Identity provides a comprehensive, process-driven, customizable, cloud-enabled, scalable, and highly available identity management solution for businesses and organizations. It provides overarching, risk-based identity and access governance functionality seamlessly integrated with automated provisioning. Functionality includes lifecycle management for users and roles, cross-platform and rule-based real-time provisioning, web-based self-service functions for users, delegated administration, request workflows, access certification, password management, metadirectory as well as auditing and reporting functionality.



DirX Directory

DirX Directory provides a standards-compliant, high-performance, highly available, highly reliable, highly scalable, and secure LDAP and X.500 Directory Server and LDAP Proxy with very high linear scalability. DirX Directory can serve as an identity store for employees, customers, partners, subscribers, and other IoT entities. It can also serve as a provisioning, access management and metadirectory repository, to provide a single point of access to the information within disparate and heterogeneous directories available in an enterprise network or cloud environment for user management and provisioning.



DirX Access

DirX Access is a comprehensive, cloud-ready, scalable, and highly available access management solution providing policy- and risk-based authentication, authorization based on XACML and federation for Web applications and services. DirX Access delivers single sign-on, versatile authentication including FIDO, identity federation based on SAML, OAuth and OpenID Connect, just-in-time provisioning, entitlement management and policy enforcement for applications and services in the cloud or on-premises.



DirX Audit

DirX Audit provides auditors, security compliance officers and audit administrators with analytical insight and transparency for identity and access. Based on historical identity data and recorded events from the identity and access management processes, DirX Audit allows answering the “what, when, where, who and why” questions of user access and entitlements. DirX Audit features historical views and reports on identity data, a graphical dashboard with drill-down into individual events, an analysis view for filtering, evaluating, correlating, and reviewing of identity-related events and job management for report generation.

For more information: support.dirx.solutions/about

Eviden is a registered trademark © Copyright 2026, Atos Group – All rights reserved.

Legal remarks

On the account of certain regional limitations of sales rights and service availability, we cannot guarantee that all products included in this document are available through the Eviden sales organization worldwide. Availability and packaging may vary by country and is subject to change without prior notice. Some/All of the features and products described herein may not be available locally. The information in this document contains general technical descriptions of specifications and options as well as standard and optional features which do not always have to be present in individual cases. Eviden reserves the right to modify the design, packaging, specifications and options described herein without prior notice. Please contact your local Eviden sales representative for the most current information. Note: Any technical data contained in this document may vary within defined tolerances. Original images always lose a certain amount of detail when reproduced.