

CoreMedia Digital Experience Platform 8
//Version 7.5.45-10

COREMEDIA



CoreMedia Digital Experience Platform 8 Developer Manual

COREMEDIA



CoreMedia Digital Experience Platform 8 Developer Manual

Copyright CoreMedia AG © 2015

CoreMedia AG

Ludwig-Erhard-Straße 18

20459 Hamburg

International

All rights reserved. No part of this manual or the corresponding program may be reproduced or copied in any form (print, photocopy or other process) without the written permission of CoreMedia AG.

Germany

Alle Rechte vorbehalten. CoreMedia und weitere im Text erwähnte CoreMedia Produkte sowie die entsprechenden Logos sind Marken oder eingetragene Marken der CoreMedia AG in Deutschland. Alle anderen Namen von Produkten sind Marken der jeweiligen Firmen.

Das Handbuch bzw. Teile hiervon sowie die dazugehörigen Programme dürfen in keiner Weise (Druck, Fotokopie oder sonstige Verfahren) ohne schriftliche Genehmigung der CoreMedia AG reproduziert oder vervielfältigt werden. Unberührt hiervon bleiben die gesetzlich erlaubten Nutzungsarten nach dem UrhG.

Licenses and Trademarks

All trademarks acknowledged.
07.Mar 2017

1. Preface	1
1.1. Audience	2
1.2. Typographic Conventions	3
1.3. CoreMedia Services	5
1.3.1. Registration	5
1.3.2. CoreMedia Releases	5
1.3.3. Documentation	6
1.3.4. CoreMedia Training	8
1.3.5. CoreMedia Support	9
1.4. Working with CoreMedia DXP 8	12
1.4.1. Getting Started	12
1.4.2. Getting an Overview	13
1.4.3. Working with the GUI	13
1.4.4. Operating the System	14
1.4.5. Extending the System	14
1.5. Change Chapter	17
2. Overview of CoreMedia DXP 8	18
2.1. Components and Architecture	21
2.1.1. Content Management Environment	22
2.1.2. Content Delivery Environment	24
2.1.3. Shared Components	24
2.1.4. Technologies	27
2.1.5. Communication between the Components	27
2.2. CoreMedia Blueprint Sites	29
3. Getting Started	31
3.1. Quick Start	32
3.2. Prerequisites	39
3.3. Configuration of the CoreMedia Workspace	44
3.3.1. Removing Optional Components	44
3.3.2. Configuring Maven	45
3.3.3. Configuring the Workspace	46
3.3.4. Configuring Vagrant Based Setup	50
3.3.5. Configuring Local Setup	53
3.3.6. In-Memory Replacement for MongoDB-Based Services	54
3.4. Customizing IBM WebSphere Commerce	58
3.4.1. Preparing the RAD Workspace	60
3.4.2. Copy Libraries	60
3.4.3. Configuring the Search	60
3.4.4. Extending REST Resources to BOD Mapping	65

3.4.5. Configuring the Cookie Domain	65
3.4.6. Multiple Logon for the Same User	66
3.4.7. Configuring REST Handlers	67
3.4.8. Applying Changes to the Management Center	68
3.4.9. Deploying the CoreMedia Fragment Connector	68
3.4.10. Customizing IBM WebSphere Commerce JSPs	72
3.4.11. Deploying the CoreMedia Widgets	73
3.4.12. Setting up SEO URLs for CoreMedia Pages	77
3.4.13. Event-based Commerce Cache Invalidation	78
3.4.14. Deploying the CoreMedia Catalog Data	79
3.4.15. Troubleshooting	80
3.5. Using the CoreMedia Workspace	82
3.5.1. Building the Workspace	82
3.5.2. Working With the Box	83
3.5.3. Locally Starting the Components	85
3.5.4. Developing with Apache (optional for e-Commerce)	91
3.5.5. Developing with Components and Boxes	98
3.5.6. Developing Against a Remote Environment	101
4. Blueprint Workspace for Developers	103
4.1. Concepts and Architecture	104
4.1.1. Maven Concepts	104
4.1.2. Blueprint Base Modules	107
4.1.3. Application Architecture	107
4.1.4. Structure of the Workspace	112
4.1.5. Project Extensions	114
4.1.6. Virtualization and Provisioning	119
4.2. Administration and Operation	125
4.2.1. Performing a Release	125
4.2.2. Deploying a System	127
4.2.3. Upgrade a System	137
4.2.4. Rollback a System	139
4.2.5. Troubleshooting	140
4.3. Development	141
4.3.1. Using Blueprint Base Modules	141
4.3.2. Developing with Extensions	146

4.3.3. Extending Content Types	155
4.3.4. Developing with Studio	157
4.3.5. Developing with the CAE	161
4.3.6. Customizing the CAE Feeder	165
4.3.7. Adding Common Infrastructure Compon- ents	165
4.3.8. Managing Properties in the Workspace	170
4.3.9. Configure Filtering in the Workspace	171
5. IBM WebSphere Commerce Integration	174
5.1. Commerce-led Integration Scenario	176
5.1.1. Commerce-led Integration Overview	176
5.1.2. Solutions for Same-Origin Policy Problem	177
5.1.3. Extending the Shop Context in Commerce-led Integration Scenario	181
5.1.4. Extending with Fragments	183
5.2. Content-led Integration	195
5.2.1. Content-led Integration Overview	195
5.2.2. Status Synchronization in the Content-led Integ- ration Scenario	196
5.2.3. Configuring Protocol-less Links for WCS	202
5.3. Communication	204
5.4. Connecting with an IBM WCS Shop	207
5.5. Link Building for Fragments	213
5.6. Enabling Preview of Commerce Category Pages in Stu- dio	215
5.7. Enabling Contract Based Preview	216
5.8. The e-Commerce API	219
5.9. Commerce Cache Configuration	221
5.10. Studio Integration of the IBM WebSphere Commerce Content	223
5.10.1. Catalog View in CoreMedia Studio Library	223
5.10.2. WCS Management Center Integration in Core- Media Studio	227
5.10.3. WCS Preview Support Features	227
5.10.4. Working with WCS Workspaces	230
5.10.5. Augmenting WCS Content	231
6. CoreMedia DXP 8 e-Commerce Blueprint - Functionality for Websites	244
6.1. Overview of e-Commerce Blueprint	245
6.2. Basic Content Management	248
6.2.1. Common Content Types	248

6.2.2. Adaptive Personalization Content Types	253
6.2.3. Tagging and Taxonomies	254
6.3. Website Management	264
6.3.1. Folder and User Rights Concept	264
6.3.2. Navigation and Contexts	265
6.3.3. Settings	268
6.3.4. Page Assembly	269
6.3.5. Overwriting Product Teaser Images	280
6.3.6. Content Lists	280
6.3.7. View Types	282
6.3.8. CMS Catalog	283
6.3.9. Teaser Management	286
6.3.10. Dynamic Templating	288
6.3.11. View Repositories	290
6.3.12. Client Code Delivery	291
6.3.13. Managing End User Interactions	293
6.3.14. Images	298
6.3.15. URLs	300
6.3.16. Vanity URLs	300
6.3.17. Content Visibility	301
6.3.18. Content Type Sitemap	302
6.3.19. Robots File	303
6.3.20. Sitemap	306
6.3.21. Website Search	308
6.3.22. Search Landing Pages	309
6.4. Website Development with Themes	311
6.4.1. CoreMedia Themes	311
6.4.2. Web Development Workflow	321
6.5. Localized Content Management	332
6.5.1. Concept	332
6.5.2. Administration	337
6.5.3. Development	342
6.6. Workflow Management	356
6.6.1. Publication	356
6.6.2. Predefined Translation Workflow	362
6.6.3. Deriving Sites	369
7. CoreMedia DXP 8 Brand Blueprint - Functionality for Web-sites	370
7.1. Overview	371
7.2. Website Features	377
7.3. Website Search	382

8. CoreMedia DXP 8 Editorial and Back-end Functionality	386
8.1. Studio Enhancements	387
8.1.1. Image Link List Editor	387
8.1.2. Content Chooser	388
8.1.3. Content Query Editor	390
8.1.4. Call-to-Action Button	392
8.1.5. External Date	393
8.1.6. Library	394
8.1.7. Bookmarks	395
8.1.8. External Library	395
8.1.9. External Preview	398
8.1.10. Settings for Studio	399
8.1.11. Content Creation	400
8.1.12. Create from Template	405
8.1.13. Site-specific configuration of Document Forms	407
8.1.14. Site Selection	408
8.1.15. Upload Files	408
8.1.16. Studio Preview Slider	411
8.2. CAE Enhancements	414
8.2.1. Using Dynamic Fragments in HTML Re- sponses	414
8.2.2. Image Cropping in CAE	416
8.3. Elastic Social	418
8.3.1. Configuring Elastic Social	419
8.3.2. Displaying Custom Information in Studio	422
8.3.3. Adding Custom Filters for Moderation View	424
8.3.4. Emailing	425
8.3.5. Curated transfer	426
8.3.6. Elastic Social Demo Data Generator	426
8.4. Adaptive Personalization	431
8.4.1. Key Integration Points	432
8.4.2. Adaptive Personalization Extension Mod- ules	432
8.4.3. CAE Integration	433
8.4.4. Studio Integration	437
8.5. Third-Party Integration	440
8.5.1. Optimizely	440
8.5.2. Open Street Map Integration	440
8.5.3. Google Analytics Integration	441

8.6. WebDAV Support	442
8.7. Advanced Asset Management	443
8.7.1. Product Asset Widget	444
8.7.2. Replaced Product and Category Images	446
8.7.3. Extract Image Data During Upload	449
8.7.4. Configuring Asset Management	451
8.7.5. Using the Adobe Drive Connector	458
9. Appendix	462
9.1. Port Reference	463
9.2. Typical LiveContext Deployment	467
9.3. Linux / Unix Installation Layout	468
9.4. IBM WebSphere Commerce REST Services used by Core- Media	470
9.5. Maven Profile Reference	475
9.6. Content Type Model	476
9.7. Link Format	478
9.8. Predefined Users	484
9.9. Database Users	487
9.10. Cookies	488
Glossary	489
Index	496

List of Figures

2.1. CoreMedia Studio with content from the IBM WebSphere Commerce system	19
2.2. System Overview	22
3.1. URLs of virtualized environment	37
4.1. Workspace Structure	112
4.2. CoreMedia Extensions Overview	115
4.3. Component Mapping	116
4.4. The new sample studio plugin	159
4.5. The sample studio plugin with plugin class and descriptor	160
5.1. The CoreMedia Perfect Chef site with dynamic price information from the IBM WebSphere Commerce shop	175
5.2. Commerce-led integration scenario	176
5.3. The Perfect Chef header as a fragment for the Aurora shop	177
5.4. Cross Domain Scripting with Fragments	178
5.5. The <code>CrossDomainEnabler</code>	179
5.6. Cross Site Scripting with fragments	180
5.7. Connection via placement name	184
5.8. CoreMedia Widgets in Commerce Composer	185
5.9. Content-led integration scenario	195
5.10. Content-led integration scenario with cookies	197
5.11. Content-led integration scenario	198
5.12. Content-led integration scenario	200
5.13. Content-led/Commerce-led scenario communication	204
5.14. Example of a Commerce API Request	205
5.15. Example of a Fragment Connector Request	206
5.16. Edit Commerce Contracts in Test Persona	216
5.17. Preview Augmented Page no Test Persona	217
5.18. Preview Augmented Page with Contracts in Test persona	217
5.19. Library with catalog in the tree view	224
5.20. Open Product in tab	225
5.21. Product in tab preview	225
5.22. Open Category in tab	226
5.23. Category in tab preview	226
5.24. Management Center in Studio	227
5.25. Time based preview affects also the <i>IBM WCS</i> preview	228
5.26. Test Persona with Commerce Customer Segments	229

5.27. Edit Commerce Segments in Test Persona	229
5.28. Workspaces selector in User Preferences Dialog	231
5.29. Catalog structure in the catalog root content item	233
5.30. Choosing a page layout for a shop page	234
5.31. Category overview page with CMS content	235
5.32. Decision diagram	236
5.33. Product detail page with CMS content highlighted by the red border	237
5.34. Page grid for PDPs	238
5.35. Product detail page with CMS assets	239
5.36. Example: Contact Us Pagegrid	240
5.37. Example: Navigation Settings for a simple SEO Page	241
5.38. Example: Navigation Settings for a custom non SEO Form	242
5.39. Special Case: Navigation Settings for the Homepage	243
6.1. Aurora category page for different devices: desktop, tablet, mobile	246
6.2. Perfect Chef homepage for different devices: desktop, tablet, mobile	247
6.3. Dynamic list of articles tagged with "Vegetables"	255
6.4. Taxonomy Administration Editor	257
6.5. Taxonomy Property Editor	258
6.6. Taxonomy Studio Settings	259
6.7. Navigation in the Perfect Chef Site	266
6.8. Breadcrumb in the Corporate Blueprint Site	266
6.9. The page grid editor	272
6.10. The main placement of a page	272
6.11. An inheriting placement	273
6.12. A locked placement	273
6.13. The layout chooser combo box	274
6.14. Teaser collection with prices	281
6.15. Layout Variant selector	283
6.16. CMS Catalog Settings	285
6.17. Default view and teaser view of an Article	287
6.18. Content Type Sitemap	303
6.19. <code>Robots.txt</code> settings	304
6.20. Channel settings with configuration for <code>Robots.txt</code> as a linked setting on a root page	305
6.21. Selection of a sitemap setup	307
6.22. Themes in the Library	312
6.23. CAE flow in detail	321

6.24. Workflow in detail	322
6.25. Linking a theme to site root	330
6.26. Multi-Site Interdependence	336
6.27. Locales Administration in CoreMedia Studio	338
6.28. Derive Site: Setting site manager group	341
6.29. Site Indicator: Setting site manager group	342
7.1. Corporate detail page for different devices	372
7.2. Teasable page with customized call-to-action button	373
7.3. Different teasers on the Brand homepage	374
7.4. Define gaps for pages	377
7.5. Setting content for collection with gap	378
7.6. SearchConfiguration Settings document	382
8.1. Image link list	388
8.2. Content chooser	389
8.3. Content Query Editor	392
8.4. Call-to-Action-Button editor	392
8.5. Call-to-Action button in teaser view	393
8.6. Externally displayed date editor	393
8.7. Setting an external date	393
8.8. Image Gallery Creation Button	394
8.9. Image Gallery Creation Dialog	394
8.10. Library List View	395
8.11. Bookmarks	395
8.12. External library showing RSS feed items	396
8.13. External Preview Dialog	398
8.14. External Preview Login	399
8.15. New content menu on the favorites toolbar	400
8.16. New content dialog	400
8.17. New content dialog for pages	401
8.18. New content dialog as button on a link list toolbar	402
8.19. New content dialog menu on a link list toolbar	402
8.20. Create from template dialog	406
8.21. The site selector on the preference tab	408
8.22. The upload files dialog	409
8.23. The slider of the Studio Preview	411
8.24. Conditions in Personalized Content and User Segment documents	437
8.25. Defining artificial context properties using Personas	438
8.26. Selecting Personas to test Personalized Content and User Segment documents	439
8.27. Example for a Open Street Map integration in a website	440

8.28. Overview over asset management part	444
8.29. Product image gallery delivered by the CMS	445
8.30. Assign a product to a picture	446
8.31. Define Product Image URLs in Management Center	447
8.32. Screenshot from Adobe Photoshop for a Picture containing XMP Data	450
8.33. Picture linked to XMP Product Reference	450
8.34. Configuration of the download portal	457
8.35. Taxonomy for assets	458
9.1. Deployment and communication overview	464
9.2. Typical deployment and ports of a LiveContext system	467
9.3. CoreMedia Blueprint Content Type Model - CMLocalized	476
9.4. CoreMedia Blueprint Content Type Model - CMNavigation	477
9.5. CoreMedia Blueprint Content Type Model - CMHasCon- texts	477
9.6. CoreMedia Blueprint Content Type Model - CMMedia	477
9.7. CoreMedia Blueprint Content Type Model - CMCollection	477
9.8. A basic absoluteUrlPrefixes Struct	481
9.9. A complete absoluteUrlPrefixes Struct	483
9.10. An initial absoluteUrlPrefixes Struct	483

List of Tables

1.1. Typographic conventions	3
1.2. Pictographs	3
1.3. CoreMedia manuals	6
1.4. Log files check list	10
1.5. Changes	17
3.1. Overview of minimum/recommended RAM	40
3.2. Optional modules and blueprints	44
3.3. Database Settings	54
3.4. Studio Configuration Properties for In-Memory Store	56
3.5. Modules in the Workspace	91
3.6. Components of the Apache Development Setup	95
3.7. Environment properties	101
4.1. RPM deployment properties	126
4.2. <code>node.js</code> configurations	131
4.3. Content type model dependencies	142
4.4. Parameters of the settings* methods	143
4.5. Blueprint Extension Descriptors and Dependencies	151
5.1. CoreMedia Content Widget configuration options	185
5.2. CoreMedia Product Asset Widget configuration options	186
5.3. Attribute of the Include tag	187
5.4. Supported usages of the externalRef attribute	189
5.5. Fragment handler usage	192
5.6. Properties for WCS connection	208
5.7. <code>config.id</code>	209
5.8. Currency configuration	210
5.9. Currency configuration	210
5.10. Properties for B2B contract based personalization	218
5.11. <code>config.id</code>	241
6.1. Overview of Content Types for common content	249
6.2. e-Commerce Content Types	249
6.3. Overview e-Commerce Content Properties	250
6.4. Overview Common Content Properties	251
6.5. <code>CMMedia</code> Properties	252
6.6. <code>CMTaxonomy</code> Properties	256
6.7. Additional <code>CMLocTaxonomy</code> Properties	256
6.8. <code>CMLinkable</code> Properties for Tagging	257
6.9. <code>CMLinkable</code> Properties for Tagging	257
6.10. Properties of <code>CMLinkable</code> for Settings Management	268

6.11. Collection Types in CoreMedia Blueprint	281
6.12. CMS Catalog: Maven parent modules	284
6.13. Properties of <code>CMTeasable</code>	287
6.14. Properties of <code>CMTemplateSet</code>	289
6.15. Client Code - Properties of <code>CMAbstractCode</code>	291
6.16. Client Code - Properties of <code>CMNavigation</code>	292
6.17. Properties for Visibility Restriction	302
6.18. Suggested Users and Groups for multi-site	339
6.19. Properties of the Site Model	343
6.20. Placeholders for Site Model Configuration	345
6.21. Example for server export and import for multi-site	350
6.22. XLIFF Properties	352
6.23. Publishing documents: actions and effects	358
6.24. Publishing folders: actions and effects	359
6.25. Predefined publication workflow definitions	361
6.26. Predefined publication workflow steps	361
6.27. User options.	362
6.28. Attributes of <code>GetDerivedContentsAction</code>	365
6.29. Attributes of <code>GetSiteManagerGroupAction</code>	366
6.30. Attributes of <code>ExtractPerformerAction</code>	366
6.31. Attributes of <code>CompleteTranslationAction</code>	367
6.32. Attributes of <code>RollbackTranslationAction</code>	368
7.1. Brand website search settings	382
7.2. Page Grid Indexing Spring Properties	384
8.1. Image Thumbnail selection rules	387
8.2. Database Settings	397
8.3. Upload Settings	410
8.4. Root Channel Context Settings	419
8.5. Context Settings for Every Channel	420
8.6. Mail Templates	425
8.7. Elastic Social Demo Data Generator operations	428
8.8. Elastic Social Demo Data Generator configuration	428
8.9. Elastic Social Demo Data Generator statistics	430
8.10. Adaptive Personalization's main Maven module in detail	433
8.11. Adaptive Personalization contexts configured for CoreMedia Blueprint	434
8.12. Predefined <code>SearchFunctions</code> in <i>CoreMedia Blueprint</i>	435
8.13. Settings for Open Street Map Integration	441
8.14. Path segments in the image URL	447
9.1. Component Port Prefix	465
9.2. Protocol / Service Port Suffix	465

- 9.3. Third-Party Services 465
- 9.4. Default Package Layout 468
- 9.5. Maven profiles 475
- 9.6. CapBlobHandler 478
- 9.7. CodeHandler 478
- 9.8. ExternalLinkHandler 478
- 9.9. PageActionHandler 478
- 9.10. PageHandler 479
- 9.11. PreviewHandler 479
- 9.12. StaticUrlHandler 479
- 9.13. TransformedBlobHandler 479
- 9.14. Global groups 484
- 9.15. Global users 484
- 9.16. Site specific groups e-Commerce 485
- 9.17. Site specific users e-Commerce 485
- 9.18. Site specific groups Brand web presence 485
- 9.19. Site specific users Brand web presence 486
- 9.20. Database Users 487

List of Examples

3.1. host entries	52
3.2. New Solr field	63
3.3. New CM_SEO_TOKEN Solr field	64
3.4. wc-dataload.xml	74
3.5. Default link setting	80
3.6. Adding Environments in <code>settings.xml</code>	102
3.7. Activating Environment in <code>settings.xml</code>	102
4.1. Dependencies for a CoreMedia application	105
4.2. Including <code>logback-common.xml</code>	109
4.3. Setting an environment property in <code>web.xml</code>	110
4.4. Setting an environment property in the context configuration	110
4.5. Enabling an Extension	117
4.6. Module structure of the extension	118
4.7. Define the component	118
4.8. Module structure with BOM POM	118
4.9. BOM POM with dependencies on submodules	118
4.10. Enabling the extension in the root POM file	118
4.11. <code>Vagrantfile</code> Example	121
4.12. Snapshot Profile	126
4.13. <code>maven-release-plugin</code>	126
4.14. An example <code>solo.rb</code> file	129
4.15. An example <code>node.json</code> file	129
4.16. <code>base.rb</code> for CentOS 6	132
4.17. <code>management.rb</code>	133
4.18. <code>replication.rb</code>	133
4.19. YUM repository	134
4.20. upgrading to a specific version	137
4.21. Yum info	138
4.22. The Spring Bean Definition for the Map of Settings Finder	144
4.23. Adding Custom Settings Finder	144
4.24. Business Logic API	145
4.25. Settings Address Adapter	145
4.26. Address Proxy	145
4.27. Activation of an Extension in the project's root POM	147
4.28. Remove CoreMedia Elastic Social Extension	149
4.29. Remove CoreMedia Adaptive Personalization Extension	149

4.30. Example for Adaptive Personalization Content in Blueprint	150
4.31. Remove CoreMedia Livecontext Extension	150
4.32. Remove CoreMedia Corporate Extension	150
4.33. Remove CoreMedia Product Asset Management Extension	151
4.34. POM file of a new Studio module	158
4.35. Maven Dependency for Logging	166
4.36. Logback Configuration	166
4.37. Change Log Directory in Tomcat	167
4.38. Automatically reload configuration file every 30 seconds	167
4.39. Dependency for JMX	168
4.40. Register the MBeans	168
4.41. Use Tomcat remote connector server	169
4.42. Use Tomcat remote connector server with authentication	169
4.43. Adding the Base Component	170
5.1. ContextProvider interface method	182
5.2. Access the Shop Context in CAE via Context API	183
5.3. Default fragment handler order	192
5.4. IBM WCS configuration in application.properties	207
6.1. Pagegrid example definition	276
6.2. A robots.txt file	303
6.3. robots.txt file generated by the example settings	306
6.4. A sitemap file	306
6.5. A sitemap index file	306
6.6. File structure of a theme	312
6.7. Theme descriptor example	313
6.8. CSS code that follows the style guide	316
6.9. Folder structure of the Saas files	317
6.10. Save selector in variable	317
6.11. Disable auto-escaping with the cm.unescape plugin	319
6.12. Example of a fallback in Freemarker	320
6.13. Difference between JSP and Freemarker type-hinting comment	320
6.14. Passing parameters	320
6.15. Theme paths in tomcat-context.xml	325
6.16. Building the theme with Grunt or Maven	327
6.17. Configuration for webresources plugin	328
6.18. Multi-Site Folder Structure Example	335

6.19. Site Folder Structure Example	335
6.20. XML of locale Struct	337
6.21. SiteModel in <code>editor.xml</code>	347
6.22. Versioned Master Link in <code>editor.xml</code>	347
6.23. CMLocalized	348
6.24. CMTeasable	348
6.25. CMSettings	349
6.26. XLIFF fragment	351
6.27. Usage of <code>GetDerivedContentsAction</code>	365
6.28. Usage of <code>GetSiteManagerGroupAction</code>	366
6.29. Usage of <code>ExtractPerformerAction</code>	367
6.30. Usage of <code>CompleteTranslationAction</code>	367
6.31. Usage of <code>RollbackTranslationAction</code>	369
8.1. Using the content query editor	390
8.2. Add content creation dialog to link list with <code>quickCreateLink-</code> <code>ListMenu</code>	402
8.3. Predicate Example	414
8.4. Predicate Customizer Example	415
8.5. Dynamic Include Link Scheme Example	415
8.6. Dynamic Include Handler Example	416
8.7. Root Channel Context Settings	420
8.8. Context Settings for Every Channel	421
8.9. Rendition Publication Configuration	455
8.10. Adding certificates to truststore	461
9.1. Configuration of URL prefix type	481

1. Preface

This manual contains the basic knowledge you should have when you want to develop with *CoreMedia Digital Experience Platform 8*. It describes the basic features and concepts of the development workspace, of the IBM WebSphere Commerce integration and of the Blueprint features.

- [Chapter 2, Overview of CoreMedia DXP 8 \[18\]](#) gives you an overview over the modules, functions and architecture of *CoreMedia Digital Experience Platform 8*.
- [Chapter 3, Getting Started \[31\]](#) shows you step by step how to install and start the components using the *Blueprint* workspace.
- [Chapter 4, Blueprint Workspace for Developers \[103\]](#) explains in depth the concepts and patterns of the *Blueprint* workspace. You will learn how to release and deploy the system and how to develop in the workspace.
- [Chapter 5, IBM WebSphere Commerce Integration \[174\]](#) describes the concepts and architecture of the integration with IBM WebSphere Commerce. You will learn how to do the configuration and how to develop with *e-Commerce API*.
- [Chapter 6, CoreMedia DXP 8 e-Commerce Blueprint - Functionality for Websites \[244\]](#) explains the content types of *CoreMedia Digital Experience Platform 8* and the website that builds on top.
- [Chapter 7, CoreMedia DXP 8 Brand Blueprint - Functionality for Websites \[370\]](#) describes the features of the *Brand Blueprint*.
- [Chapter 8, CoreMedia DXP 8 Editorial and Back-end Functionality \[386\]](#) describes the extensions of *CoreMedia Blueprint* to the standard system.
- [Chapter 9, Appendix \[462\]](#) contains reference information, such as the tag library, ports, the content type model or Maven profiles.

1.1 Audience

This manual is intended for architects and developers who want to work with *CoreMedia Digital Experience Platform 8* or who want to learn about the concepts of the product. The reader should be familiar with *CoreMedia CMS*, *IBM WebSphere Commerce*, *Spring*, *Maven* and *Chef*.

1.2 Typographic Conventions

CoreMedia uses different fonts and types in order to label different elements. The following table lists typographic conventions for this documentation:

Element	Typographic format	Example
Source code	Courier new	<code>cm systeminfo start</code>
Command line entries		
Parameter and values		
Class and method names		
Packages and modules		
Menu names and entries	Bold, linked with	Open the menu entry Format Normal
Field names	Italic	Enter in the field <i>Heading</i>
CoreMedia Components		The <i>CoreMedia Component</i>
Applications		Use <i>Chef</i>
Entries	In quotation marks	Enter "On"
(Simultaneously) pressed keys	Bracketed in "<>", linked with "+"	Press the keys <Ctrl>+<A>
Emphasis	Italic	It is <i>not</i> saved
Buttons	Bold, with square brackets	Click on the [OK] button
Code lines in code examples which continue in the next line	\	<code>cm systeminfo \ -u user</code>
Mention of other manuals		See the [Studio Developer Manual] for more information.

Table 1.1. Typographic conventions

In addition, these symbols can mark single paragraphs:




Pictograph	Description
	Tip: This denotes a best practice or a recommendation.
	Warning: Please pay special attention to the text.

Table 1.2. Pictographs

Pictograph	Description
	Danger: The violation of these rules causes severe damage.

1.3 CoreMedia Services

This section describes the CoreMedia services that support you in running a CoreMedia system successfully. You will find all the URLs that guide you to the right places. For most of the services you need a CoreMedia account. See [Section 1.3.1, “Registration” \[5\]](#) for details on how to register.

CoreMedia User Orientation for CoreMedia Developers and Partners

Find the latest overview of all CoreMedia services and further references at:

<http://documentation.coremedia.com/new-user-orientation>



- [Section 1.3.1, “Registration” \[5\]](#) describes how to register for the usage of the services.
- [Section 1.3.2, “CoreMedia Releases” \[5\]](#) describes where to find the download of the software.
- [Section 1.3.3, “Documentation” \[6\]](#) describes the CoreMedia documentation. This includes an overview of the manuals and the URL where to find the documentation.
- [Section 1.3.4, “CoreMedia Training” \[8\]](#) describes CoreMedia training. This includes the training calendar, the curriculum and certification information.
- [Section 1.3.5, “CoreMedia Support” \[9\]](#) describes the CoreMedia support.

1.3.1 Registration

In order to use CoreMedia services you need to register. Please, start your [initial registration via the CoreMedia website](#). Afterwards, contact the CoreMedia Support (see [Section 1.3.5, “CoreMedia Support” \[9\]](#)) by email to request further access depending on your customer, partner or freelancer status so that you can use the CoreMedia services.

1.3.2 CoreMedia Releases

Downloading and Upgrading the Blueprint Workspace

CoreMedia provides its software as a Maven based workspace. You can download the current workspace or older releases via the following URL:

<http://releases.coremedia.com/dxp8>

Refer to our [Blueprint Github mirror repository](#) for recommendations to upgrade the workspace either via Git or patch files.



If you encounter a 404 error then you are probably not logged in at GitHub or do not have sufficient permissions yet. See [Section 1.3.1, “Registration” \[5\]](#) for details about the registration process. If the problems persist, try clearing your browser cache and cookies.

Maven artifacts

CoreMedia provides its release artifacts via Maven under the following URL:

<https://repository.coremedia.com>

You have to add your CoreMedia credentials to your Maven settings file as described in section [CoreMedia Digital Experience Platform 8 Developer Manual \[i\]](#).

License files

You need license files to run the CoreMedia system. Contact the support (see [Section 1.3.5, “CoreMedia Support” \[9\]](#)) to get your licences.

1.3.3 Documentation

CoreMedia provides extensive manuals and Javadoc as PDF files and as online documentation at the following URL:

<http://documentation.coremedia.com/dxp8>

The manuals have the following content and use cases:

Manual	Audience	Content
CoreMedia Utilized Open-Source Software	Developers, architects, administrators	This manual lists the third-party software used by CoreMedia and lists, when required, the licence texts.
Supported Environments	Developers, architects, administrators	This document lists the third-party environments with which you can use the CoreMedia system, Java versions or operation systems for example.
Studio User Manual, English	Editors	This manual describes the usage of <i>CoreMedia Studio</i> for editorial and administrative work. It also describes the usage of the <i>Adaptive Personalization</i> and <i>Elastic Social</i> GUI that are integrated into <i>Studio</i> .

Table 1.3. CoreMedia manuals

Manual	Audience	Content
LiveContext for IBM WebSphere Manual	Developers, architects, administrators	<p>This manual gives an overview over the structure and features of CoreMedia LiveContext. It describes the integration with the IBM WebSphere Commerce system, the content type model, the <i>Studio</i> extensions, folder and user rights concept and many more details. It also describes administrative tasks for the features.</p> <p>It also describes the concepts and usage of the project workspace in which you develop your CoreMedia extensions. You will find a description of the Maven structure, the virtualization concept, learn how to perform a release and many more.</p>
Operations Basics Manual	Developers, administrators	This manual describes some overall concepts such as the communication between the components, how to set up secure connections, how to start application or the usage of the watchdog component.
Adaptive Personalization Manual	Developers, architects, administrators	This manual describes the configuration of and development with <i>Adaptive Personalization</i> , the CoreMedia module for personalized websites. You will learn how to configure the GUI used in <i>CoreMedia Studio</i> , how to use predefined contexts and how to develop your own extensions.
Analytics Connectors Manual	Developers, architects, administrators	This manual describes how you can connect your CoreMedia website with external analytic services, such as Google Analytics.
Content Application Developer Manual	Developers, architects	This manual describes concepts and development of the <i>Content Application Engine (CAE)</i> . You will learn how to write JSP or Freemarker templates that access the other CoreMedia modules and use the sophisticated caching mechanisms of the CAE.
Content Server Manual	Developers, architects, administrators	This manual describes the concepts and administration of the main CoreMedia component, the <i>Content Server</i> . You will learn about the content type model which lies at the heart of a CoreMedia system, about user and rights management, database configuration, and more.

Manual	Audience	Content
Elastic Social Manual	Developers, architects, administrators	This manual describes the concepts and administration of the <i>Elastic Social</i> module and how you can integrate it into your websites.
Importer Manual	Developers, architects	This manual describes the structure of the internal CoreMedia XML format used for storing data, how you set up an <i>Importer</i> application and how you define the transformations that convert your content into CoreMedia content.
Search Manual	Developers, architects, administrators	This manual describes the configuration and customization of the <i>CoreMedia Search Engine</i> and the two feeder applications: the <i>Content Feeder</i> and the <i>CAE Feeder</i> .
Site Manager Developer Manual	Developers, architects, administrators	This manual describes the configuration and customization of <i>Site Manager</i> , the Java based stand-alone application for administrative tasks. You will learn how to configure the <i>Site Manager</i> with property files and XML files and how to develop your own extensions using the <i>Site Manager API</i> .
Studio Developer Manual	Developers, architects	This manual describes the concepts and extension of <i>CoreMedia Studio</i> . You will learn about the underlying concepts, how to use the development environment and how to customize <i>Studio</i> to your needs.
Unified API Developer Manual	Developers, architects	This manual describes the concepts and usage of the <i>CoreMedia Unified API</i> , which is the recommended API for most applications. This includes access to the content repository, the workflow repository and the user repository.
Workflow Manual	Developers, architects, administrators	This manual describes the <i>Workflow Server</i> . This includes the administration of the server, the development of workflows using the XML language and the development of extensions.

If you have comments or questions about CoreMedia's manuals, contact the Documentation department:

Email: documentation@coremedia.com

1.3.4 CoreMedia Training

CoreMedia's training department provides you with the training for your CoreMedia projects either in the CoreMedia training center or at your own location.

You will find information about the CoreMedia training program, the training schedule and the CoreMedia certification program at the following URL:

<http://www.coremedia.com/training>

Contact the Training department at the following email address:

Email: training@coremedia.com

1.3.5 CoreMedia Support

CoreMedia's support is located in Hamburg and accepts your support requests between 9 am and 6 pm MET. If you have subscribed to 24/7 support, you can always reach the support using the phone number provided to you.

To submit a support ticket, track your submitted tickets or receive access to our forums visit the CoreMedia Online Support at:

<http://support.coremedia.com/>

Do not forget to request further access via email after your initial registration as described in [Section 1.3.1, "Registration" \[5\]](#). The support email address is:

Email: support@coremedia.com

Create a support request

CoreMedia systems are distributed systems that have a rather complex structure. This includes, for example, databases, hardware, operating systems, drivers, virtual machines, class libraries and customized code in many different combinations. That's why CoreMedia needs detailed information about the environment for a support case. In order to track down your problem, provide the following information:

Support request

- Which CoreMedia component(s) did the problem occur with (include the release number)?
- Which database is in use (version, drivers)?
- Which operating system(s) is/are in use?
- Which Java environment is in use?
- Which customizations have been implemented?
- A full description of the problem (as detailed as possible)
- Can the error be reproduced? If yes, give a description please.
- How are the security settings (firewall)?

In addition, log files are the most valuable source of information.

To put it in a nutshell, CoreMedia needs:

Support checklist

- 1. a person in charge (ideally, the CoreMedia system administrator)
- 2. extensive and sufficient system specifications
- 3. detailed error description
- 4. log files for the affected component(s)
- 5. if required, system files

An essential feature for the CoreMedia system administration is the output log of Java processes and CoreMedia components. They're often the only source of information for error tracking and solving. All protocolling services should run at the highest log level that is possible in the system context. For a fast breakdown, you should be logging at debug level. The location where component log output is written is specified in its `< appName>-logback.xml` file.

Log files

Which Log File?

Mostly at least two CoreMedia components are involved in errors. In most cases, the *Content Server* log files in `coremedia.log` files together with the log file from the client. If you are able locate the problem exactly, solving the problem becomes much easier.

Where do I Find the Log Files?

By default, log files can be found in the CoreMedia component's installation directory in `/var/logs` or for web applications in the `logs/` directory of the servlet container. See the "Logging" chapter of the [Operations Basics Manual] for details.

Component	Problem	Log files
CoreMedia Studio	general	CoreMedia-Studio.log coremedia.log
CoreMedia Editor	general	editor.log coremedia.log workflowserver.log capclient.properties
	check-in/check-out	editor.log coremedia.log workflowserver.log capclient.properties
	publication or pre-view	coremedia.log (Content Management Server) coremedia.log (Master Live Server)

Table 1.4. Log files check list

Component	Problem	Log files
		workflowserver.log capclient.properties
	import	importer.log coremedia.log capclient.properties
	workflow	editor.log workflow.log coremedia.log capclient.properties
	spell check	editor.log MS Office version details coremedia.log
	licenses	coremedia.log (Content Management Server) coremedia.log (Master Live Server)
Server and client	communication errors	editor.log coremedia.log (Content Management Server) coremedia.log (Master Live Server) *.jpif files
	preview not running	coremedia.log (content server) preview.log
	website not running	coremedia.log (Content Management Server) coremedia.log (Master Live Server) coremedia.log (Replication Live Server) Blueprint.log capclient.properties license.zip
Server	not starting	coremedia.log (Content Management Server) coremedia.log (Master Live Server) coremedia.log (Replication Live Server) capclient.properties license.zip

1.4 Working with CoreMedia DXP 8

This chapter guides you to the download area, other manuals and training courses depending on your skills and the tasks you want to accomplish. CoreMedia documentation is organized in such a way, that each component manual contains all required information for the configuration, operation and development of the component. Only the user manuals for editors and other users are in separate documents.

Chapters and sections that have only a noun in the title usually contain conceptual information while a title with an "-ing" indicates an instructional chapter.

1.4.1 Getting Started

To start with *CoreMedia DXP 8* you should open the following address in your browser:

<http://releases.coremedia.com/dxp8>

Here, you will find a short quick start description and links to all resources for *CoreMedia DXP 8*. You can download *CoreMedia Blueprint*.

Implementing *CoreMedia CMS* typically means to start a project with configuration and customization tasks. Therefore, *CoreMedia DXP 8* is designed for the developer. It simply integrates with a common Maven based development environment. By default, it supports deployment via RPM or ZIP files but it's up to the developers to decide how to actually deploy *CoreMedia DXP 8*. With *CoreMedia Blueprint*, CoreMedia delivers a blueprint for development and deployment.

- Read the Supported Environments document available at <https://documentation.coremedia.com/dxp8/supported-environments> to learn which databases, browsers, operation systems, Java versions, Portal version and servlet container are supported by *CoreMedia DXP 8*.
- Read the [CoreMedia Digital Experience Platform 8 Developer Manual] to learn how to install CoreMedia components with *CoreMedia Blueprint*.
- Read the [CoreMedia Digital Experience Platform 8 Developer Manual] to learn about the *Blueprint* features.
- Read the [CoreMedia Operations Basics] manual to learn basic operation tasks.
- Read the [Third-Party Licenses Manual] if you want to know which open source software is used by *CoreMedia DXP 8*.
- Attend the "Operations" training at the CoreMedia Training Center, see <http://www.coremedia.com/training-schedule/> for the current schedule.

1.4.2 Getting an Overview

If you want to get familiar with the concepts and coverage of *CoreMedia Digital Experience Platform 8*, then this manual is the starting point. Nevertheless, it only gives you a rough insight. If you want to learn more about all the components that comprise *CoreMedia DXP 8* you should read the following chapters:

- Read the "CoreMedia Content Server" chapter in the [Content Server Manual] to learn something about the basic component of the CoreMedia system.
- Read the "Overview" chapter in the manual of every component you are interested in.
- Attend the "Fundamentals" training at the CoreMedia Training Center, see <http://www.coremedia.com/training-schedule/> for the current schedule.

1.4.3 Working with the GUI

CoreMedia DXP 8 comes with different GUIs that support different tasks, such as managing content and user generated content or personalize the output. Their usage is described in separate manuals or chapters shown below. All these manuals are intended for editors and other non-technical staff.

CoreMedia Studio

CoreMedia Studio is the editor tool for all users. It is web based and requires no installation. Its easy-to-use interface with instant preview and form based editing makes content creation easier than ever. All other CoreMedia components integrate their GUI into *CoreMedia Studio*. Create new content, access your IBM WebSphere content, manage your website or user generated content or publish new content to your customers.

- Read the [Studio User Manual] for details.

Elastic Social

The *Elastic Social* GUI is integrated with *CoreMedia Studio*.

- Read the "Working with User Generated Content" chapter of the *CoreMedia Studio User Manual* for details

Adaptive Personalization Management

CoreMedia Adaptive Personalization comes with a management GUI that bases on the same technology as *CoreMedia Studio*. It lets you define selection rules, test user profiles and user segments.

- ➔ Read the "Using the Adaptive Personalization GUI" chapter of the [CoreMedia Studio User Manual] for details.

CoreMedia Site Manager

The *CoreMedia Site Manager* is a management component for power users and administrators. Manage rights and users.

- ➔ Read the [CoreMedia Operations Basics Manual] for details on user management.

1.4.4 Operating the System

The components of *CoreMedia Digital Experience Platform 8* are configured using property files and you can use JMX to manage them. In addition, *CoreMedia DXP 8* contains tools to monitor the status of its components. The following chapters are intended for operators and administrators but developers should read the chapters as well.

- ➔ Read the [CoreMedia Operations Basics Manual] for some operational concepts and tasks.
- ➔ Each component manual contains a configuration chapter. Read this chapter if you want to learn details about a component's configuration.
- ➔ Attend the *CoreMedia Deployment* and *CoreMedia Operations* training.

1.4.5 Extending the System

CoreMedia Digital Experience Platform 8 is a very flexible software system, that you can adapt to all your needs. It integrates nicely with a Maven based development environment. CoreMedia is shipped with manuals that cover general development concepts such as the workspace and the *Unified API* and with manuals that cover the development with specific components.

General Concepts

- ➔ Read the [CoreMedia Digital Experience Platform 8 Developer Manual] to learn how to develop extensions using *Blueprint* workspace.
- ➔ Read the [Unified API Developer Manual] in order to learn how to use the most fundamental CoreMedia API.
- ➔ Read the [Content Server Manual] in order to learn how to define your own content types.

Developing editorial components

If you want to develop components for editorial purpose, you might refer to one of the following manuals:

- Read the [Studio Developer Manual] in order to learn how to extend *CoreMedia Studio*.
- Read the [Site Manager Developer Manual] in order to learn how to extend the *Site Manager*.
- Read the [Unified API Developer Manual] in order to learn how to develop client applications from the scratch accessing the *CoreMedia CMS* via the *Unified API*.
- Attend the *CoreMedia Studio Customization* training in order to learn how to extend *CoreMedia Studio*.
- Attend the *CoreMedia Site Manager Customization* training in order to learn how to extend the *CoreMedia Site Manager*.
- Attend the *Client Development* training in order to learn the usage of the *CoreMedia Unified API*.

Developing workflows

CoreMedia CMS contains a customizable *Workflow Server* that you can adapt to your needs. *CoreMedia CMS* is delivered with workflows that support publishing tasks, but the *Workflow Server* can support much more complicated processes.

- Read the [Workflow Manual] in order to learn how to define your own workflows.
- Attend the *Workflow Implementation* training in order to define your own workflows. Learn how to use the powerful *CoreMedia Workflow XML* format and develop your own classes in order to extend the *Workflow Server* functionality.

Developing websites

CoreMedia CMS is a web content management system and its main purpose is to deliver content to various devices. Not only to a PC but to all gadgets such as mobile phones or tablet PCs.

- Read the [Content Application Developer Manual] in order to learn how to develop fast, dynamic websites that support sophisticated caching. Learn how to use the *CAE*.
- Read the [Elastic Social Manual] in order to learn how to extend your websites with user generated content, such as comments or ratings.
- Read the [Adaptive Personalization Manual] in order to learn how to deliver personalized content.
- Read the [Search Engine Manual] in order to learn how to make your websites searchable.
- Attend the Web Application Development (WAD) training, in order to get hands-on experience in the development of *CAE* applications.
- Attend the *Advanced Web Application Development (AWAD)* training, in order to get an in-depth look into the workings of the *CAE*. Learn about controllers, link schemes, caching and more.

- ➔ Attend the *Caching* training in order to get familiar with the subtleties of caching with the CAE.

1.5 Change Chapter

In this chapter you will find a table with all major changes made in this manual.

Table 1.5. Changes

Section	Version	Description
Chapter 5, IBM WebSphere Commerce Integration [174]	7.5.44	Reorders chapter
Chapter 9, Appendix [462]	7.5.44	Adds overview of cookies delivered by the CMS system
Chapter 9, Appendix [462]	7.5.43	Adds typical LiveContext deployment
Section 6.4, “Website Development with Themes” [311]	7.5.41	New description of web development with themes
Chapter 3, Getting Started [31]	7.5.41	Split subchapters about configuration of IBM WebSphere Commerce and CoreMedia DXP 8 into two separate chapters.
Appendix - Predefined Users [484]	7.5.41	Added description of predefined users for brand web presence.
Section 7.3, “Website Search” [382]	7.5.32	Added description of <i>Brand Blueprint</i> Website Search, its configuration and feeding of pages with content from their page grids.
Chapter 7, CoreMedia DXP 8 Brand Blueprint - Functionality for Websites [370]	7.5.31	Added chapter about <i>Brand Blueprint</i> .
Section 3.4.9, “Deploying the CoreMedia Fragment Connector” [68]	7.5.19	Added description of <code>connectionTimeout</code> and <code>socketTimeout</code> for fragment connector.
Section 5.1.4, “Extending with Fragments” [183]	7.5.19	Added description on how to add CMS content to product detail pages and category overview pages.
Section “Locales Administration” [337]	7.5.8	Added description for supranational locale definition

2. Overview of CoreMedia DXP 8

CoreMedia Digital Experience Platform 8 is the next-generation experience management platform from CoreMedia that lets you build highly engaging, multi-channel branded e-Commerce experiences as well as corporate sites for your global customers.

Now you can easily bridge the gap between a pure e-Commerce system which is focused on the more transactional aspects of the buying process and content-driven brand sites that focus on engaging user experiences.

CoreMedia Studio allows your business users to efficiently create and manage engaging digital experiences across the customer journey by enriching the basic product information with storytelling by adding editorial content and media assets from the CoreMedia CMS. You can seamlessly blend catalog content and CMS content to any degree and on any delivery channel - and ensure brand-consistency through multi-language and multi-site localization tools.

The *CoreMedia Digital Experience Platform 8* platform bundles all components to help you manage every aspect of your blended digital experiences from content to commerce:

- CoreMedia CMS platform
- CoreMedia Studio
- CoreMedia Blueprints for e-Commerce and corporate sites
- CoreMedia e-Commerce Bridge for IBM WebSphere Commerce
- CoreMedia Site Manager
- CoreMedia Elastic Social
- CoreMedia Adaptive Personalization
- CoreMedia Advanced Asset Management

CoreMedia Digital Experience Platform 8 was designed to empower your team in creating and managing highly relevant and engaging experiences for your customers from a single, easy-to-use business user interface. Customers should always get the information they need, independent of the device they use or the time they connect - delivered in an optimized fashion for the current customer's context.

CoreMedia Studio allows business users to create and manage experiences based on context and to define and test rules and user segments for personalization in real-time. Content can be easily mixed with e-Commerce catalog items. Editors can intuitively select the products and categories from the catalog and place them on the site just as they are accustomed from other web content.

CoreMedia Digital Experience Platform 8 ships with CoreMedia Blueprints for e-Commerce and corporate sites that provide a high-level of prefabrication of common features and use-cases. The source code is provided for easy customization to your specific needs for competitive differentiation.

Built upon industry-leading best practices with a fully responsive and adaptive mobile first design plus a wealth of ready-to-use layout modules, your development team can jump-start on a strong foundation proven in many customer projects whilst retaining full flexibility. A predefined Maven based development environment is provided.

Leveraging the CoreMedia CAE technology, you can dynamically and contextually combine relevant content from CoreMedia CMS, CoreMedia Elastic Social and your e-Commerce system and deliver the combined experience in real-time on all channels with utmost performance using the sophisticated caching.

Elastic Social allows your end users to contribute user-generated content such as product reviews, comments and ratings - whilst providing an intuitive moderation interface to your business users that also allows for editorial re-purposing of user-generated content.

CoreMedia Site Manager is the administration console for user and rights management.



Figure 2.1. CoreMedia Studio with content from the IBM WebSphere Commerce system

[Section 2.1, “Components and Architecture” \[21\]](#) describes *CoreMedia DXP 8* in more detail:

2.1 Components and Architecture

CoreMedia DXP 8 has been developed to provide a universal solution for the creation and management of content.

The use of modern development tools and open interfaces enables the system to be flexibly adapted to enterprise requirements. For this purpose, worldwide standards for information processing, such as XML, HTML, HTTP, REST, Ajax, WebDAV, CORBA and the Java Platform are used or supported.

CoreMedia Digital Experience Platform 8 is a distributed system, that consists of several components for different use cases.

- *CoreMedia Content Server*
 - Content Management Server
 - Master Live Server
 - Replication Live Server
- *CoreMedia Workflow Server*
- *CoreMedia Content Application Engine*
- *CoreMedia Importer*
- *CoreMedia Search Engine*
 - CoreMedia Content Feeder
 - CoreMedia CAE Feeder
- *CoreMedia e-Commerce Bridge for IBM WebSphere Commerce*
- *CoreMedia Studio*
- *CoreMedia User Changes web application*
- *CoreMedia Site Manager*
- *CoreMedia Elastic Social*
- *CoreMedia Adaptive Personalization*
- *CoreMedia Advanced Asset Management*
- *CoreMedia Blueprints*

In addition, *CoreMedia DXP 8* relies on some third-party systems:

- An IBM WebSphere Commerce Server for e-Commerce
- A relational database to store the content and user data
- A MongoDB NoSQL database to store the user generated content

- ➔ An LDAP server for user management
- ➔ Servlet containers to run the web applications

Conceptually, a CoreMedia system can be divided into the *Content Management Environment* where editors create and manage the content and the *Content Delivery Environment* where the content is delivered to the customers. Some components are used in both environments, mostly to give you a realistic preview of your websites. [Figure 2.2, “System Overview” \[22\]](#) provides an overview of a *CoreMedia DXP 8* system with all components installed:

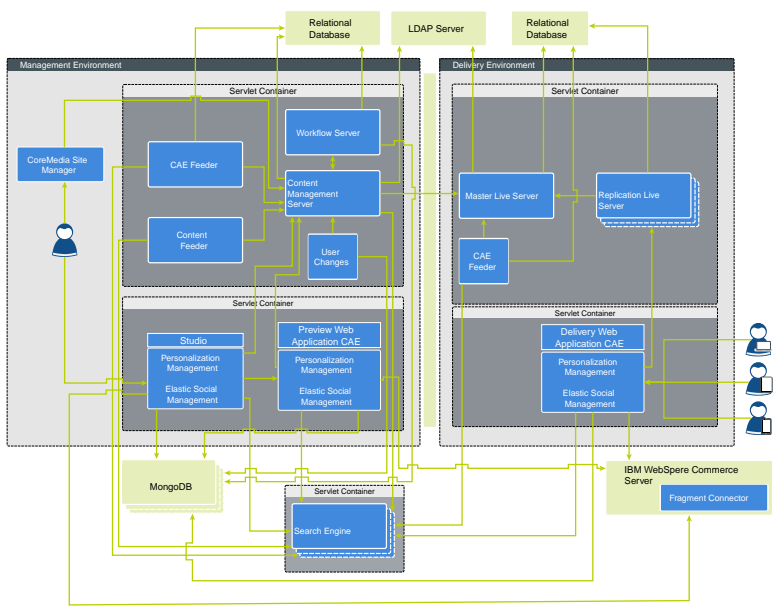


Figure 2.2. System Overview

The following sections describe in short the aim of all components, some main technologies used in *CoreMedia DXP 8* and give a short overview over the communication between the components.

2.1.1 Content Management Environment

The Content Management Environment is the place where you create and manage your website with the *Content Management Server* and *Studio* at its heart. A freely adaptable content model allows you to manage and deliver every type of digital content including text, video, images, music and many more.

The following components are solely located in the Content Management Environment:

CoreMedia Content Management Server

The *Content Management Server* is a web application that manages the content in *CoreMedia DXP 8*.

CoreMedia Studio

Studio is a web application. It integrates the complete workflow used by online editors from the creation, over management to preview publication of digital experiences with contextual content. *Studio* is a web application that bases on modern standards such as Ajax. Therefore, it can be used like a common desktop application; fast, reliable but without installation. *Studio* integrates the *CoreMedia Adaptive Personalization* and *Elastic Social* GUI and has an integrated preview window where you can see your content in its context. You can even see the effects of personalization or time-dependent publication.

CoreMedia Studio lets you access the content of the IBM WebSphere Commerce Server and integrates its management console. Content can be mixed easily with e-Commerce catalog items. Editors can intuitively select the items from the catalog and place them on the site just as they are accustomed from other web contents.

CoreMedia User Changes web application

The *CoreMedia User Changes* web application is a listener, which shows the current work of the logged-in editor in *Studio*. This web application supports the functionality of Control Room in *Studio*.

CoreMedia Site Manager

CoreMedia Site Manager is a Java based rich client for administrators. It offers additional functionality like user and rights management.

CoreMedia Importer

You can use the Importer to import content from external sources into the management system. A freely adaptable importer framework based on JAXP is used to build content sets and pipelines and to invoke content transformations, using XSL, DOM and Streams.

CoreMedia Workflow Server

The *CoreMedia Workflow Server* is a web application that executes and manages workflows. *CoreMedia DXP 8* comes with predefined workflows for publication and translation, but you can also define your own workflows.

CoreMedia Content Feeder

The *Content Feeder* is a web application that collects the content from the *Content Management Server* and delivers it to the *Search Engine* for indexing. Thus, the

Content Feeder is necessary to make content searchable in *Studio* for the editor. The *Content Feeder* listens for changes in the content and triggers the indexing of the changed or newly created content.

2.1.2 Content Delivery Environment

The Content Delivery Environment of *CoreMedia CMS* may consist of the *Master Live Server*, several *Replication Live Servers* (which are optional), the *CoreMedia CAE*, *CoreMedia Elastic Social*, the *Search Engine* and *Adaptive Personalization*. It manages the approved and published online data and adds user generated content.

CoreMedia Master Live Server

The *Master Live Server* manages the *CoreMedia* repository in the Content Delivery Environment. It receives this content from the *Content Management Server* during publication. The *Content Application Engine* fetches the content from the *Master Live Server* or from the *Replication Live Servers*.

CoreMedia Replication Live Server

The optional *Replication Live Servers* replicate the content of the *Master Live Server* in order to enhance reliability and scalable performance.

2.1.3 Shared Components

Some components of *CoreMedia DXP 8* are used in both environments. The *e-Commerce Bridge*, for example, is used in the Management Environment to manage content from the IBM WebSphere Commerce system in *Studio* and in the Delivery Environment to include content from the IBM WebSphere Commerce system in the pages generated by *Content Application Engine*. Other components, like the *Content Application Engine*, are used to provide the editor with a preview of the live site.

CoreMedia e-Commerce Bridge for IBM WebSphere Commerce

CoreMedia e-Commerce Bridge for IBM WebSphere Commerce connects the *CoreMedia CMS* with the IBM WebSphere Commerce Server (WCS). It provides functionality to read catalog items, such as products or marketing spots, and to display them on web pages. You can also display price information and availability of products on the site. All e-Commerce functions are provided by an e-Commerce Java API that enables you to extend your shop application.

The e-Commerce bridge also enables you to enrich pages rendered by the IBM WebSphere Commerce system with content delivered by the CAE of *CoreMedia DXP 8*. This way, you can enhance your shop pages with more engaging content.

Finally, the *CoreMedia e-Commerce Bridge for IBM WebSphere Commerce* synchronizes user sessions between the IBM WebSphere Commerce system and the CoreMedia system, so that users only have to sign in once.

CoreMedia Content Application Engine (CAE)

The *CoreMedia Content Application Engine* represents a stack for building client applications with *CoreMedia CMS*. It is a web application framework which allows fast development of highly dynamic, supportable and personalizable applications and websites. Sophisticated caching mechanisms allows for dynamic delivery even in high-load scenarios with automatic invalidation of changed content.

The *CoreMedia Content Application Engine* combines content from all CoreMedia components, from your e-Commerce system and other third-party systems in so-called content beans and delivers the content to your customers in all formats. The preview in *Studio* and the website visited by your customers is delivered by the CAE

CoreMedia Search Engine

A *CoreMedia CMS* system comes with Apache Solr as the default search engine, which can be used from the editors on content management site and from the applications on content delivery site. The editor, for example, can perform a fast full text search in the complete repository. The pluggable search engine API allows you to use other search engines than Apache Solr for the website search.

CoreMedia CAE Feeder

The *CAE Feeder* makes content beans searchable by sending their data to the *Search Engine* for indexing.

CoreMedia Adaptive Personalization

CoreMedia Adaptive Personalization enables enterprises to deliver the most appropriate content to users depending on the 'context' – the interaction between the user, the device, the environment and the content itself. *CoreMedia Adaptive Personalization* is a powerful personalization tool. Through a series of steps it can identify relevant content for individuals. It can draw on a user's profile, IBM WebSphere Commerce segment, preferences and even social network behavior. Use *CoreMedia Adaptive Personalization* to deliver highly relevant and personalized content to users, at any given moment in time.

The GUI is integrated into *CoreMedia Studio* for easy creation and testing of user segments and selection rules.

CoreMedia Elastic Social

CoreMedia Elastic Social enables enterprises to engage with users, entering a conversation with them and stimulating discussion between them. Use *Elastic Social* to enable Web 2.0 functionality for Web pages and start a vibrant community. It offers all the features it takes to build a community – personal profiles, preferences, relationships, ratings and comments. *CoreMedia Elastic Social* is fully customizable to reflect the environment you want to create, and offers unlimited horizontal scalability to grow with the community and your business vision. It also integrates with *CoreMedia Studio* so you can manage comments and external users right from your common workplace.

CoreMedia Advanced Asset Management

CoreMedia Advanced Asset Management is a module that adds asset management functionality to the system. Digital assets, such as images or documents, and their licences can be managed in *CoreMedia Studio*. From an asset, you can create common content items that can be used in the IBM WCS system.

CoreMedia Blueprint

For a quick start, *CoreMedia DXP 8* is delivered with two fully customizable blueprint applications including best practices and example integration of available features. *CoreMedia Blueprint* contains a ready-made content model for navigation and multi-language support. It contains for instance solutions for e-Commerce items, taxonomy, rating, integration with web analytics software and user created page layouts. *CoreMedia Blueprint* comes as a Maven based workspace for development.

The workspace is the result of *CoreMedia*'s long year experience in customer projects. As *CoreMedia CMS* is a highly customizable product adaptable to your specific needs, the first thing you used to do when you started to work with *CoreMedia CMS* was to create a proper development environment on your own. *CoreMedia Digital Experience Platform 8* addresses this challenge with a reference project in a pre-defined working environment that integrates all *CoreMedia* components and is ready for start.

CoreMedia Blueprint workspace provides you with an environment which is strictly based on today's de facto standard for managing and building Java projects by using Maven. That way, building your project artifacts is a matter of simply executing `mvn clean install`. Developers are able to test all the various *CoreMedia CMS* components directly within the same environment by executing `mvn tomcat7:run` and `mvn tomcat7:run-war` respectively. No further deployment is necessary.

Maven based environment

With the introduction of Chef as the provision tool of choice and Vagrant as the tool to prepare VirtualBox virtualized environments, setting up the server backend to start developing with *Blueprint* workspace is now a matter of minutes, rather than hours. A simple `vagrant up` will provide you with all databases, content

repository and search services without the need to install and configure any project specific software on the developer's system.

To achieve this simplicity all components within the *Blueprint* workspace are pre-configured with hostnames, ports, database schemes, users and passwords such that you don't need to worry about configuration while developing within the *Blueprint* workspace. Instead, right from the start, you may concentrate on the real work, on the business logic that creates your company's value in the first place.

Blueprint workspace creates RPM or Zip artifacts out of the box, which you can use for deployment. You can preconfigure or post-configure your components that means at build time or at installation time, respectively.

For details on each component, please refer to the individual manuals. Online documentation for all these components is available online at <http://documentation.coremedia.com/cm7>.

Using a virtualized development infrastructure

2.1.4 Technologies

The following technologies are featured by *CoreMedia DXP 8*:

WebDAV

CoreMedia CMS supports WebDAV. This allows you to edit content with WebDAV-enabled applications such as Photoshop, GoLive, etc.

WebDAV, Web-based Distributed Authoring and Versioning, is an IETF standard set of platform-independent extensions to HTTP that allows users to collaboratively edit and manage files on remote Web servers. WebDAV features XML properties on metadata, locking - which prevents authors from overwriting each other's changes - namespace manipulation and remote file management.

LDAP

The *CoreMedia CMS* supports LDAP server for user management.

Lightweight Directory Access Protocol (LDAP) is a set of protocols for accessing information directories. It is based on the standards within the X.500 standard, but is significantly simpler. Unlike X.500, LDAP supports TCP/IP, which is necessary for any type of Internet access. Because it's a simpler version of X.500, LDAP is sometimes called X.500-lite.

2.1.5 Communication between the Components

Communication between the individual components on both the production side and the *Live Server* is performed via CORBA and HTTP. MongoDB uses the Mongo Wire Protocol. The *Production* and *Live Systems* can be secured with a Firewall if

the servers are located on different computers. The servers contact the databases over a JDBC interface,

CoreMedia DXP 8 and IBM WebSphere communicate over REST interfaces. The concrete communication differs slightly based on the selected deployment scenario which are the *content-led scenario* and the *commerce-led scenario*. The following picture shows the communication how fragments of a page are delivered from the CAE and from the commerce system.

Processing

On the production side of the CoreMedia system, content is created and edited with *CoreMedia Studio*, with custom clients or imported by the importers. Once editing or import of contents is completed, they are approved and published via the *CoreMedia Workflow*. During the publication process, the content is put online onto the *Master Live Server*. If available, *Replication Live Servers* get noticed and reproduce the changes. Then the content is put online by the *Replication Live Server*. User generated content is produced via *Elastic Social* and is stored in MongoDB. Editors can use the *Studio* plugin to moderate this content.

Content from the IBM WebSphere Commerce Server is not copied into the CoreMedia system. Instead, references to the content are hold and are resolved when content is delivered.

The *CoreMedia CAE* in combination with *Adaptive Personalization* and *Elastic Social* creates dynamic HTML pages or any other format (XML, PDF, etc.) from the internal and external content and CoreMedia templates.

2.2 CoreMedia Blueprint Sites

CoreMedia DXP 8 Experience Platform contains Brand Blueprint and e-Commerce Blueprint for a quick start. They come with four different sites that support different use cases.

Aurora Augmentation (en)

This site belongs to the e-Commerce Blueprint. It is intended for a company that wants to extend their IBM WebSphere Commerce B2C online shop with engaging assets and content from the CoreMedia system, the so called commerce-led scenario (see [Section 5.1, “Commerce-led Integration Scenario” \[176\]](#)). Editors can add inspiring content from the CMS such as images, videos, articles to the standard WCS pages. They do not need to enter the WCS, but can use *CoreMedia Studio* for their work, taking advantage of the sophisticated preview of *Studio*.

Perfect Chef (en/de)

This site belongs to the e-Commerce Blueprint. It is intended for a company that wants to create a content-driven e-Commerce brand experience in the CoreMedia CMS, the so-called content-led scenario (see [Section 5.2.1, “Content-led Integration Overview” \[195\]](#)). Please note, that product detail pages and the checkout process are delivered by the WCS.

Aurora B2B Augmentation (en)

This site belongs to the e-Commerce Blueprint. It is intended for a company that wants to extend their IBM WebSphere Commerce B2B online shop with engaging content from the CoreMedia system, the so called content-led scenario (see [Section 5.1, “Commerce-led Integration Scenario” \[176\]](#)). This site only works with FEP8. The use case is similar to the one of the Aurora Augmentation site for B2C shops. In addition, the *Studio* preview supports contracts, so that a test persona will only see content related to products from the WCS which is allowed by their contract.

Chef Corp. Site (en/de)

This site belongs to the Brand Blueprint. It is intended for a company that wants to offer their corporate site as an engaging experience for its users on all devices with a fully responsive design. The site contains no e-Commerce shop, but the company can use the CoreMedia catalog to manage and present their products on the web site.

Removing Sites

You can remove sites and features that you do not need from your workspace. To remove the Perfect Chef and Aurora sites, remove the LiveContext extension as described in [Section “Removing the e-Commerce Blueprint” \[150\]](#). To remove the Brand Site, remove the corporate extension as described in [Section “Removing the Brand Blueprint” \[150\]](#).

3. Getting Started

This chapter describes the installation, configuration and the start of the components of *CoreMedia Digital Experience Platform 8*.

- [Section 3.1, “Quick Start” \[32\]](#) describes the fastest way to get a CoreMedia system up and running on your local machine using the new flexible deployment.
- [Section 3.2, “Prerequisites” \[39\]](#) describes the software and hardware requirements that you need to fulfill in order to work with *CoreMedia Digital Experience Platform 8*.
- [Section 3.3, “Configuration of the CoreMedia Workspace” \[44\]](#) describes how you have to configure the workspace and the virtualized setup in order to start developing with *CoreMedia Digital Experience Platform 8*.
- [Section 3.4, “Customizing IBM WebSphere Commerce” \[58\]](#) describes how you have to configure the IBM WebSphere system.
- [Section 3.5, “Using the CoreMedia Workspace” \[82\]](#) describes how you can build the workspace and start the component in order to start developing with a running system.

The CoreMedia workspace contains two blueprints, the e-Commerce Blueprint and the Brand Blueprint. Both blueprints can be used together as demonstrated in the example sites (see [Section 2.2, “CoreMedia Blueprint Sites” \[29\]](#)). However, you can also use both blueprints separately. Deactivate the not-used blueprint as described in [Section 4.3.2, “Developing with Extensions” \[146\]](#). If you do not want to use the e-Commerce Blueprint you can skip [Section 3.4, “Customizing IBM WebSphere Commerce” \[58\]](#).

Unless specified otherwise command line examples are given in Unix style. The path to the root of the *Blueprint* workspace directory will be referenced by the variable `$CM_BLUEPRINT_HOME`.



3.1 Quick Start

CoreMedia Digital Experience Platform 8 is a content management system for the developer. You do not get a program to install and run, but a workspace to develop within, to build and to deploy artifacts from. The workspace uses Maven for the build and contains a virtualized environment that makes it easy to build and run the components on your local machine. See [Chapter 2, Overview of CoreMedia DXP 8 \[18\]](#) for an overview.

This chapter guides you through all steps you have to perform in order to get the CoreMedia system running on your local machine. No alternative options or advanced configurations are described. The "Further Reading" section of each step contains links to additional content, but you do not need to read these chapters for the purpose of the quickstart.

What do you get?

When you are finished with all steps, you will have a virtualized CoreMedia system with all components, such as Content Management Server, Master Live Server, Studio or CAE up and running in VirtualBox on your computer. All the sample content delivered with the workspace is imported and you can use *CoreMedia Studio* to browse through the content. You will have no connection with an IBM WCS e-Commerce system.

You need Internet access to get everything up and running.



Step 1: Getting a Login for CoreMedia

Goal

You have a login to the CoreMedia software download page, the contributions Github repository, the documentation and the CoreMedia artifact repository.

Steps

1. Ask your project manager for your company's account details or contact the CoreMedia support. Keep in mind, that you have to ask explicitly for the access rights to the CoreMedia Github contributions repository. See CoreMedia's website for the contact information of the support at <http://www.coremedia.com/support>.

Check

Got to documentation.coremedia.com/dxp8 and <https://github.com/coremedia-contributions/dxp8-blueprint> and enter your credentials. You should be able to use the online documentation and see the contributions repository.

Step 2: Getting License Files for the CoreMedia System

Goal

You have licenses for the CoreMedia system.

Steps

Ask your project manager, your key account manager or your partner manager for the CoreMedia licences.

Check

You have a Zip file that contains three zipped licence files.

Further Reading

- See Section 4.6, “CoreMedia Licenses” in *CoreMedia Operations Basics* for details about the license file format.

Step 3: Checking the Hardware requirements

Goal

You are sure, that your computer meets the hardware requirements.

Steps

1. Check that your computer has at least a dual-core CPU with >2GHz and at least 12GB of RAM. Otherwise, the build will be very slow and the virtual environment cannot be started.

Further reading

- [Section 3.2, “Prerequisites” \[39\]](#) describes the hardware requirements a bit more in detail.

Step 4: Check and Install all Required Third-Party Software

Goal

All required third-party software (such as Java, Git, Maven, Chef...) is installed on your computer and has the right version.

Steps

1. Open the supported environments document at <https://documentation.core-media.com/dxp8/supported-environments-en.pdf> and check that you have installed the right version of Java and that you have the right OS. The `JAVA_HOME` variable must be set.

2. Check that you have Maven 3.3.3 installed.
3. Check that VirtualBox 5.0.16 and Vagrant 1.8.1 is installed, otherwise install in this respective order.
4. With **chef -v**, check that the Chef Development Kit 0.12.0 is installed, otherwise install ChefDK.
5. Install the *vagrant-omnibus*, *vagrant-ohai*, *nugrant* and *vagrant-berksshelf* Vagrant plugins. You will find the required versions in the `Vagrantfile` file in the main directory of the *CoreMedia Blueprint* workspace. Use the following call and replace `<pluginName>` and `<pluginVersion>` with the name as written in this paragraph and with the version, respectively

```
vagrant plugin install <pluginName> --plugin-version  
"=<pluginVersion>"
```

Further reading

- [Section 3.2, “Prerequisites” \[39\]](#) describes the required software in more detail.
- [Section 3.3.4, “Configuring Vagrant Based Setup” \[50\]](#) gives more details about the installation of the virtualization components.

Step 5: Cloning the Workspace

Goal

You have the *CoreMedia Blueprint* workspace on your hard disk.

Steps

1. Got to <https://github.com/coremedia-contributions/dxp8-blueprint> and get the GitHub clone URL.
2. On your local machine, clone the repository using Git:

```
git clone <GitHub Repo URL>
```

3. In the cloned repository, get a list of all tags:

```
git tag
```

4. Create your working branch from the tag you want to use as your starting point:

```
git checkout -b <yourBranchName> <tagName>
```

Check

The Git `clone` command has succeeded.

Further reading

- [Chapter 4, *Blueprint Workspace for Developers* \[103\]](#) describes the structure of the workspace, the concepts behind the workspace and how you can work with the workspace.
- [Section 3.3.3, “Configuring the Workspace” \[46\]](#) describes further configuration of the workspace which is required for development and deployment.
- On <https://releases.coremedia.com/dxp8/overview/> click the link to the latest download to find a description on how to download a specific release tag.

Step 6: Move the Licenses to the Right Location

Goal

The CoreMedia license files are at the right location of your workspace.

Steps

1. Unzip the license file to a temporary folder.
2. Copy and rename the Zip file which contains `_CS_` in its name to `<Blueprint Workspace>/modules/server/content-management-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`
3. Copy and rename the Zip file which contains `_MLS_` in its name to `<Blueprint Workspace>/modules/server/master-live-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`
4. Copy and rename the Zip file which contains `_RLS_` in its name to `<Blueprint Workspace>/modules/server/replication-live-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`

Check

The files are at the right location. Later, the Content Server in the virtualized environment runs without problems.

Further reading

- [Section 3.3.3, “Configuring the Workspace” \[46\]](#) describes other ways to provide the license files.

Step 7: Configure the Repository Settings and Check Maven Configuration

Goal

Your Maven `settings.xml` file contains the settings required to connect with the CoreMedia Nexus repository.

Steps

1. Follow the steps described in [Section 3.3.2, “Configuring Maven” \[45\]](#).

Check

When you build the workspace, all artifacts are found.

Step 8: Building the Workspace

Goal

The workspace has been build, so that the development setup can be started. The build takes some time. On an Intel i7 processor with 16GB RAM around 20 minutes.

Steps

1. In the main directory of the workspace call:

```
mvn clean install -DskipTests
```

Check

The Maven build ends with message "Build successful".

Further reading

- [Section 3.5.1, “Building the Workspace” \[82\]](#) describes more options for the Maven build.
- [Section 3.3.1, “Removing Optional Components” \[44\]](#) describes how you can remove parts of the workspace that you do not need.
- [Section 3.3.6, “In-Memory Replacement for MongoDB-Based Services” \[54\]](#) describes how you can replace MongoDB for Studio services with an in-memory solution.

Step 9: Starting the Development Environment

Goal

You have a VirtualBox in which all CoreMedia components are running. All CoreMedia sample sites with sample content can be used. However, since no WCS system is connected you will see no e-Commerce content.

The first start takes some time, because a lot of data will be downloaded. On an Intel i7 processor with 16GB RAM and a fast Internet connection around 20 minutes.

Steps

1. In the main directory of the workspace call `vagrant up`.
2. Confirm the dialogs that open up.

Check

You see a list of URLs in the console where you called `vagrant up`.

```
--> blueprint:
--> blueprint: ## Delivery:
--> blueprint:
--> blueprint: * Live PerfectChef via [xip.io](http://helios.192.168.252.100.xip.io) or by [alias](http://helios.blueprint-box)
--> blueprint:
--> blueprint: * Live Corporate via [xip.io](http://corporate.192.168.252.100.xip.io) or by [alias](http://corporate.blueprint-box)
--> blueprint: * JMX : 'service:jmx:rmi://blueprint-box:49098/jndi/rmi://blueprint-box:49099/jmxrmi'
--> blueprint:
--> blueprint: ## Solr Master:
--> blueprint:
--> blueprint: * [Admin UI](http://blueprint-box:44080/solr/)
--> blueprint:
--> blueprint: * JMX : 'service:jmx:rmi://blueprint-box:44098/jndi/rmi://blueprint-box:44099/jmxrmi'
--> blueprint:
--> blueprint: ## Solr Slave:
--> blueprint:
--> blueprint: * [Admin UI](http://blueprint-box:45080/solr/)
--> blueprint:
--> blueprint: * JMX : 'service:jmx:rmi://blueprint-box:45098/jndi/rmi://blueprint-box:45099/jmxrmi'
--> blueprint:
--> blueprint: ## MongoDB:
--> blueprint:
--> blueprint: * [REPT](http://blueprint-box:28017)
```

Figure 3.1. URLs of virtualized environment

Further reading

- [Section 2.2, “CoreMedia Blueprint Sites” \[29\]](#) gives a short overview over the sample sites.
- [Section 3.5.2, “Working With the Box” \[83\]](#) describes in more detail how you can work with VirtualBox and vagrant.
- [Section 4.1.6, “Virtualization and Provisioning” \[119\]](#) describes the concepts of virtualization in *CoreMedia DXP 8*.

Step 10: Logging into the Virtualized Studio

Goal

You are logged into *Studio* and you can edit content.

Steps

1. Enter the following URL into your browser:

```
https://studio-helios.192.168.252.100.xip.io
```

2. You get a message, that the connection is not secure. Add an exception and proceed.
3. Log in with user "Rick" and password "Rick".

You will get an error message, because no WCS system is connected with the CoreMedia system. However, you can use *Studio* with CoreMedia content.

Check

You see the *Studio* UI and can edit content.

Further reading

- [Appendix - Predefined Users \[484\]](#) shows the predefined users of *CoreMedia Blueprint*.
- Read *CoreMedia Studio User Manual* to learn how to work with *CoreMedia Studio*.

Step 11: Starting a Local Studio

Goal

You have a locally running *Studio* that is connected with the *Content Server* in the virtualized environment.

Steps

1. In the *CoreMedia Blueprint* workspace go to `modules/studio/studio-webapp`
2. Enter the following Maven command:

```
mvn tomcat7:run -Pvagrant
```

Check

You can log in at <http://localhost:40080> as user "Rick" with password "Rick".

Further reading

- [Section 3.5.3, "Locally Starting the Components" \[85\]](#) describes how you can start all components directly on your local machine.

3.2 Prerequisites

In order to work with the *Blueprint* workspace you need to meet some requirements.

Path length limitation in Windows

The *CoreMedia Blueprint* workspace contains long paths and deeply nested folders. If you install the *CoreMedia Blueprint* workspace in a Windows environment, keep the installation path shorter than 25 characters. Otherwise, unzipping the workspace might fail or might lead to missing files due to the 260 bytes path limit of Windows.



Account

In order to get access to the download page, to the CoreMedia contributions repository and to CoreMedia's Maven repository (<https://repository.coremedia.com>), you need to have a CoreMedia account. You can obtain an account from [CoreMedia Support](#).

Getting the CoreMedia Workspace

You can use Git to clone the *CoreMedia DXP 8* workspace via:

→ <https://github.com/coremedia-contributions/dxp8-blueprint>

Or you can download a Zip file of a specific tag:

→ <https://github.com/coremedia-contributions/dxp8-blueprint/releases>

Find the current online documentation at:

→ <http://documentation.coremedia.com/dxp8>

You will also find the download links at the CoreMedia release page at <http://releases.coremedia.com/dxp8>.

Hardware

- At least a dual-core CPU with 2GHz, a quad-core CPU is recommended, because *CoreMedia CMS* code makes heavy use of multithreading.
- The minimum RAM you need is 8GB which is enough if your locally tested components are connected to remote test environments. If the system is started by vagrant locally you will need at least 12GB of RAM. In both scenarios 16 GB are recommended.

- The operating system must provide adequate resources to the components. At least 5000 processes and 25000 file handles should be available initially. You should then monitor the system to tune these settings, because the number and the types of deployed components vary greatly. In Unix environments the `ulimit` command can be used to configure resource limits for individual users.

Development Scenario	Minimum	Recommended
Using Remote System	8GB	16GB
Using CMS Components started in Vagrant	12GB	16GB

Table 3.1. Overview of minimum/recommended RAM

Required Software

- A supported Java SDK (see [Supported Environments](#)). The variable `JAVA_HOME` must be set.
- A supported browser (see [Supported Environments](#))
- [Maven](#) 3.3.3
- An IDE. CoreMedia suggests IntelliJ Idea because it has the best support for *CoreMedia Studio* development.
- If you want to build the workspace with tests, you need a PhantomJS 2.1.1 installation on your computer that is added to your path.
- CoreMedia license files for starting the various *Content Servers*. If you do not already have the files, request your licenses from the CoreMedia [support](#).

Security-Enhanced Linux (SELinux)

By default, the preconfigured CoreMedia test system deployment does not support SELinux. If you want to use SELinux, you have to adapt the setup on your own as part of further hardening efforts for your production environment.



Developer Setup

In the developer setup, all components are running in VirtualBox, including the required databases, but not the WCS.

CentOS / Redhat Enterprise Linux 6 only



The original vagrant based setup deployment approach is supported on CentOS / Redhat Enterprise Linux 6.7 only. Thus all related configuration files in the *Blueprint* workspace are suitable for Apache 2.2 and SysV init by default.

Please refer to `deployment/chef/README.md` for more information regarding the new flexible deployment approach supporting CentOS / Redhat Enterprise Linux 7.2 as well (including Apache 2.4 and systemd init).

See also <https://atlas.hashicorp.com/coremedia>.

- [VirtualBox](#) 5.0.16
- [Vagrant](#) 1.8.1
- [Chef Development Kit](#) 0.12.0

The ChefDK is the easiest way to get started with *Chef*. It defines a common workflow for cookbook development, including dedicated tooling like [Kitchen](#) and [Berkshelf](#).

```
$ chef -v
Chef Development Kit Version: 0.12.0
chef-client version: 12.8.1
berks version: 4.3.0
kitchen version: 1.6.0
```

Please note that the preconfigured [Chef Client](#) in our Vagrant test setup is 12.8.1.

A dedicated [Chef Server](#) 12.6.0 is recommended for production deployments. Especially to handle sensitive secrets like database passwords, SSL certificates or API tokens. Please refer to, for example, [chef-vault](#) as a solid option beyond [Encrypted Data Bags via a shared secret](#).

A dedicated private [Chef Supermarket](#) server 2.5.2 or higher is recommended to improve your Chef Cookbook release process (optional).

Further installation and configuration requirements for Vagrant and VirtualBox can be found in [Section 3.3.4, “Configuring Vagrant Based Setup” \[50\]](#).

Non-Virtualized Setup

In the non-virtualized setup, you have to install the required databases locally and also start all CoreMedia components on your computer.

- A supported local database listening on the default port (see [Supported Environments](#)).
- A local MongoDB database listening on the default port (see [Supported Environments](#)).

Additional Software for e-Commerce Blueprint only



An IBM WebSphere Commerce 7.0 Feature Pack 7 / Fix Pack 9 system with installed Interim Fixes (IFixes) for the following Authorized Program Analysis Reports (APAR)

FEP7

→ JR55049.fep ("Cumulative Interim Fix for WebSphere Commerce Version 7 Fix Pack 9")

→ JR52306.fep (Mandatory cumulative interim fix for WebSphere Commerce Version 7 Feature Pack 7")



An IBM WebSphere Commerce 7.0 Feature Pack 8 / Fix Pack 9 system with installed Interim Fixes (IFixes) for the following Authorized Program Analysis Reports (APAR)

FEP8

→ JR56287.fep and JR56287.fep ("Cumulative Interim Fix for WebSphere Commerce Version 7")

→ JR56662 ("JR56217 regression issue")

→ JR57043 ("BCS error due to token created for Store 0")

Additional Software



A repository manager such as [Nexus](#) or [JFrog Artifactory](#). For evaluation purposes you can build the *Blueprint* workspace against the original repositories, but if you start your real project with multiple developers you will need a repository manager.

For an overview of exact versions of the supported software environments please refer to the Supported Environments document at [CoreMedia Online Documentation](#).

Internet access

CoreMedia provides the *CoreMedia Digital Experience Platform 8* components as Maven artifacts. These components in turn depend on many third-party components. If your operator has not yet set up and populated a local repository manager, you need Internet access so that Maven can download the artifacts.

Maven and Internet access

The *CoreMedia Blueprint* workspace relies heavily on Maven as a build tool. That is, Maven will download CoreMedia artifacts, third-party components and Maven plugins from the private CoreMedia repository and other, public repositories. This might interfere with your company's internet policy. Moreover, if a big project accesses public repositories too frequently, the repository operator might



block your domain in order to prevent overload. The best way to circumvent both problems is to use a repository manager like [Sonatype Nexus](#), that decouples the development computers from direct Internet access.

3.3 Configuration of the CoreMedia Workspace

Before you can start with development, you have to do some configurations, in part, depending on the Blueprint you want to work with.

- ➔ Remove the blueprints and modules that you do not want to use as described in [Section 3.3.1, “Removing Optional Components” \[44\]](#) .
- ➔ Adapt your Maven settings to the required repositories as described in [Section 3.3.2, “Configuring Maven” \[45\]](#).
- ➔ Adapt the workspace to your own project as described in [Section 3.3.3, “Configuring the Workspace” \[46\]](#). Configure, for example, groupId, version, deployment repositories and CoreMedia licenses.
- ➔ If you want to develop with the virtualized server environment, which is the recommended way, then you have to do some network configuration as described in [Section 3.3.4, “Configuring Vagrant Based Setup” \[50\]](#).
- ➔ If you do not want to use the virtualized environment, then you have to do some database configuration and host mapping as described in [Section 3.3.5, “Configuring Local Setup” \[53\]](#).

3.3.1 Removing Optional Components

The *CoreMedia Digital Experience Platform 8* workspace contains a complete CoreMedia system with all the core components and optional modules which have to be licensed separately. See [Section 2.1, “Components and Architecture” \[21\]](#) for an overview of all components. Before you start with development, remove all modules that you do not need.

Name	Description	Removal
CoreMedia Adaptive Personalization	Module for the work with personalized content and personas.	See Section “Removing the Adaptive Personalization Extension” [149]
CoreMedia Elastic Social	Module for the work with external users and user generated content, such as ratings or comments.	See Section “Removing the Elastic Social Extension” [149]
Advanced Asset Management	Module for the work with assets, such as images or documents.	See Section “Removing the Advanced Asset Management Extensions” [151]
e-Commerce Blueprint	Blueprint for the integration with an IBM WebSphere Commerce system.	See Section “Removing the e-Commerce Blueprint” [150]

Table 3.2. Optional modules and blueprints

Name	Description	Removal
Brand Blueprint	Blueprint for a brand website with responsive templates.	See Section “Removing the Brand Blueprint” [150]

3.3.2 Configuring Maven

CoreMedia strongly recommends to use a repository manager to mirror CoreMedia's Maven repository, for example [Sonatype Nexus](#). Alternatively, if a repository manager is not available, configure your credentials for the CoreMedia Maven repositories in your `~/.m2/settings.xml` file as shown below. Simply replace `USERNAME` and `PASSWORD` with your CoreMedia user name and password. It is strongly recommended, that you do not enter the password in plaintext in the `settings.xml` file but encrypt the password. To do so, follow the instructions at <http://maven.apache.org/guides/mini/guide-encryption.html> or any other available Maven documentation.

```
<settings>
  <mirrors>
    <!--
      | <mirror>
      |   <id>central-mirror</id>
      |   <mirrorOf>central</mirrorOf>
      |   <name>mirror of maven central</name>
      |   <url>http://my.repository.com/repo/path</url>
      | </mirror>
      | <mirror>
      |   <id>coremedia-releases-mirror</id>
      |   <mirrorOf>coremedia.external.releases</mirrorOf>
      |   <name>coremedia external releases mirror</name>
      |   <url>http://my.repository.com/repo/path</url>
      | </mirror>
    -->
  </mirrors>

  <servers>
    <server>
      <id>coremedia.external.releases</id>
      <username>USERNAME</username>
      <password>PASSWORD</password>
    </server>
    <server>
      <id>coremedia.external.livecontext.releases</id>
      <username>USERNAME</username>
      <password>PASSWORD</password>
    </server>
  </servers>

  <pluginGroups>
    <pluginGroup>com.coremedia.maven</pluginGroup>
    <pluginGroup>org.sonatype.plugins</pluginGroup>
  </pluginGroups>
</settings>
```

MAVEN_OPTS

Maven requires the following minimal memory settings:

```
MAVEN_OPTS=-Xmx2048m -XX:MaxPermSize=512m
```

The `MaxPermSize` JVM option is only required when using Java 7. Omit it when using Java 8.

3.3.3 Configuring the Workspace

The *Blueprint* workspace comes ready to use. However, there are some environment specific configurations to be adjusted at the very beginning of a project. You may skip these steps only if you are just going to explore the workspace, you will neither share your work with others nor release it, and you will start over from scratch again with your actual project.

Changing the groupId

The `groupId` of the *CoreMedia Blueprint* workspace is `com.coremedia.blueprint`. While this works from the technical point of view, you have to change it to a project specific `groupId`, because CoreMedia reserves the possibility to provide versioned artifacts of this `groupId`.

Since the `groupId` is needed to denote the parent POM file, it cannot be inherited but occurs in every `pom.xml` file. You can simply globally replace all occurrences of `<groupId>com.coremedia.blueprint</groupId>` in *Blueprint* workspace `pom.xml` files with your project `groupId`.

Selecting deployment repositories

Only the artifacts below the `$CM8_BLUEPRINT_HOME/modules` hierarchy will be deployed to a Maven repository. See [Section 4.2.1, “Performing a Release” \[125\]](#) for a deeper introduction in releasing and deploying artifacts.

To configure the repository URL to deploy the artifacts to, you need to adapt the `distributionManagement` section of the root `pom.xml` and all BOM `pom.xml` files of the extensions, for example:

```
→ $CM8_BLUEPRINT_HOME/modules/extensions/alx/alx-bom/pom.xml
→ $CM8_BLUEPRINT_HOME/modules/extensions/es/es-bom/pom.xml
→ $CM8_BLUEPRINT_HOME/modules/extensions/es-p13n/es-p13n-
  bom/pom.xml
→ $CM8_BLUEPRINT_HOME/modules/extensions/*/*-bom/pom.xml
```

CoreMedia strongly advises you to use a repository manager.

Using a Version Control System

The *CoreMedia Blueprint* workspace is developed and maintained in a Git repository, therefore the root `pom.xml` contains references to a CoreMedia internal repository in the scm tag. You need to replace this setting with your own source control URL in order to perform a release. Please refer to existing Maven documentation for details.

When you do not use a distributed VCS such as Git, for example Subversion, you have to disable local checkout in the *maven-release-plugin* (see [Release Plugin](#)). Open the root POM file of the workspace and set the `localCheckout` property to "false".



Change line endings (optional)

The *Blueprint* workspace is delivered in UNIX format (line breaks). Depending on your local setup it might be required to change the line endings to DOS format (carriage return and line break).

Configuring the licenses

Currently, there are several ways to configure license files in the workspace required for the *Content Servers*:

- Copy the license files to the server modules source tree.
- Provide license artifacts and use a license profile to add the license as a dependency.
- Use one of the approaches above for development licenses and configure an absolute path for deployment using either preconfiguration or post-configuration filtering.

These approaches are described in the next sections. The license files are delivered in a Zip file. Unpack the file to get three further Zip files. The name of the files follows this scheme:

```
<Version>_CoreMedia_AG_<ServerType>_DATA_<UniqueCode>_license.zip
```

The server type abbreviation has the following meaning:

- CS: Content Management Server
- MLS: Master Live Server
- RLS: Replication Live Server

Copying license files to source tree

Copying the license files to your source tree is the simplest and most straightforward approach. It can be achieved by simply copying the Zip files to the appropriate locations and rename them to `license.zip`:

- *Content Management Server* license to `modules/server/content-management-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`.
- *Master Live Server* license to `modules/server/master-live-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`.
- *Replication Live Server* license to `modules/server/replication-live-server-webapp/src/main/webapp/WEB-INF/properties/corem/license.zip`.

Using a license profile

If you want to use the profile approach, you need a repository manager (see [section “Additional Software” \[42\]](#)) for your project to provide the license artifacts via a secured internal repository.

The license artifacts must be of type `war` and the license should be packaged at `WEB-INF/properties/corem/license.zip` within the artifact. You should deploy the artifact with a custom `groupId` and custom release version (not a snapshot version) and should not use the values predefined in the workspace. The classifiers `production`, `publication` and `replication` refer to the type of the license and you can either keep it this way or use different GAV (`GroupId`, `ArtifactId`, `Version`) coordinates to distinguish the license type.

The default workspace profiles `internal-licenses` are defined in the files:

- `modules/server/content-management-server-webapp/pom.xml`
- `modules/server/master-live-server-webapp/pom.xml`
- `modules/server/replication-live-server-webapp/pom.xml`

and look like the following snippet:

```
<profile>
  <id>internal-licenses</id>
  <dependencies>
    <dependency>
      <groupId>YOUR PROJECTS GROUP ID</groupId>
      <artifactId>development-license</artifactId>
      <version>A RELEASE VERSION</version>
      <type>war</type>
      <classifier>production</classifier>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
</profile>
```



```
</dependencies>
</profile>
```

The profile approach has the advantage that you can define two profile types, one for development licenses and one for production licenses, if you want to deploy the license together with the artifact.

Configuring licenses for deployment

You can use different approaches for providing development and production licenses. It would be sensible, for instance to provide your production license files only as file system resources on production servers while you still provide development licenses with both approaches described before.

For production licenses as file system resources, you can configure the license location using one of the filtering approaches described in [Section 4.3.9, “Configure Filtering in the Workspace”](#) [171].

Using the preconfiguration approach, you need to configure `CMS_LICENSE`, `MLS_LICENSE` and `RLS_LICENSE` in the `packages/src/main/filters/preconfigure.properties`

Using the post-configuration approach, you need to configure the properties in the files on the target machine, for instance

- `configure.CMS_LICENSE` in the `/etc/coremedia/cm7-cms-tomcat.properties`
- `configure.MLS_LICENSE` in the `/etc/coremedia/cm7-mls-tomcat.properties`
- `configure.RLS_LICENSE` in the `/etc/coremedia/cm7-rls-tomcat.properties`

Chef configuration

To configure the license locations for the *Chef* provisioning run, you need to override the default attribute. For a chef only setup, this can be done in the role executing the server recipe, for example in the file `boxes/chef/vagrant-chef-repo/roles/management.rb` you need to add an entry to the `override_attributes` map:

```
"override_attributes" => {
  "coremedia" => {
    "configuration" => {
      "configure.CMS_LICENSE" => "/opt/coremedia/cms-license.zip"
    }
  }
}
```

For development purposes with *Vagrant*, you can provide the license via one of the shared folders for example `boxes/target/rpm-repo` and set the configura-

tion attribute in the `chef.json` hash in the `Vagrantfile`. Simply apply to it the same `coremedia` hash shown in the listing above.

3.3.4 Configuring Vagrant Based Setup

To install Vagrant and VirtualBox, download the installers with the versions described in [section “Developer Setup” \[40\]](#). Afterwards install [VirtualBox](#) first and then [Vagrant](#).

Make sure that a `HOME` environment variable is pointing to your users home directory and that the path to Virtualbox is in your `PATH` variable.



Vagrant Plugins

By default, the CoreMedia configuration depends on the following plugins that you have to install:

- The *vagrant-omnibus* plugin will install the specified *Chef Client* version. The plugins documentation can be found [here](#).
- The *vagrant-ohai* plugin will install *ohai* and set `node[ipaddress]` to the IP of the correct network adapter, that is `192.168.252.100`. The plugins documentation can be found [here](#).
- The *nugrant* plugin provides simple configuration overrides on machine, user or project level. The plugins documentation can be found [here](#).
- The *vagrant-berkshelf* plugin provides dependency management for cookbooks. Like with *Maven*, cookbook dependencies are being resolved and downloaded from a central repository. For further details visit the official *Berkshelf* documentation [here](#). The plugins documentation can be found [here](#). In general, please make sure that the *Berkshelf* version shipped with your Chef Development Kit installation is supported (refer to known issues and related workarounds at <https://github.com/berkshelf/vagrant-berkshelf/issues> before contacting CoreMedia Support).

To install the *Vagrant* plugins, simply call `vagrant plugin install PLUGIN-NAME`. For example, to prepare your virtualized environment, you need to run the following command:

```
vagrant plugin install vagrant-omnibus --plugin-version "= 1.4.1"
```

Please refer to the `Vagrantfile` for the required plugin versions.



More plugins

To further improve your development experience with *Vagrant*, you can choose from the many *vagrant* plugins, available in the *Vagrant* ecosystem.

A list of some of the most popular plugins is maintained [here](#). To reduce the turn-around time for provisioning, CoreMedia recommends one of the plugins to manage snapshots, that is, the *sahara* or the *vagrant-vbox-snapshot* plugin. With snapshot capabilities, you can reset to the last working state of a complete setup within seconds and rerun the provisioning step.

Networking

The virtual box configured with the `Vagrantfile` from this workspace is connected by a *VirtualBox* host-only adapter with the IP `192.168.252.1` and the subnet `255.255.255.0`. The box itself has the IP address `192.168.252.100`.

Windows and *VirtualBox* have issues with the creation and removal of host-only adapters. Under some circumstances, *Windows* tends to ignore the IP/Subnet configuration, given to the adapter. When that happens, *Windows* gives the adapter an APIPA (Automatic Private IP Address) address of the form `169.254.x.x` with the subnet `255.255.0.0`. This makes any box unreachable from your host machine.

To workaround this issue, you need to create a correct adapter only once and prevent *Vagrant* from destroying and recreating it all the time which might cause possible errors. To achieve that, create a virtual box definition, and register it with the correct adapter. *Vagrant* will then skip the removal of that adapter and the recreation of a new one. As a nice side effect, you won't need to click away the UAC pop-ups and do not need administrator rights anymore. The box itself neither requires any additional hard disk space, RAM or OS nor will it ever be started, it just blocks the adapter.



Host Mappings

Host mappings are required to use *CoreMedia Elastic Social*. The CoreMedia documentation provides further information about concepts and technologies of *Elastic Social*. *CoreMedia Elastic Social*'s multi-tenancy concept is bound to the domain of the particular application.

By default, the *Vagrant* setup uses [XIP IO](#), a free generic DNS service, provided by 37Signals, the founders of *Ruby on Rails*. By encoding the IP of the box within the URLs, XIP IO maps every request to the encoded IP, relieving you from modifying your `etc/host` mapping. When you start the box with `vagrant up`, you will get a list of all the URIs. The format is:

```
<name>.192.168.252.100.xip.io
for example
corporate.19.168.252.100.xip.io for the Live CAE with the
corporate site
```

If you don't want to use XIP IO, you need to adapt the configuration in the Vagrant file and add IP mappings to fake the two domains *corporate* and *helios*:

- ➔ On Linux add them to `/etc/hosts`.
- ➔ On Windows add them to `C:\Windows\System32\drivers\etc\hosts`.

Example 3.1. host entries

```
192.168.252.100    blueprint-box
192.168.252.100    studio-helios.blueprint-box
192.168.252.100    studio-corporate.blueprint-box
192.168.252.100    preview-helios.blueprint-box
192.168.252.100    preview-corporate.blueprint-box
192.168.252.100    editor.blueprint-box
192.168.252.100    webdav.blueprint-box
192.168.252.100    helios.blueprint-box corporate.blueprint-box
192.168.252.100    shop-helios.blueprint-box
192.168.252.100    shop-preview-helios.blueprint-box
192.168.252.100    shop-preview-production-helios.blueprint.box
```

Proxy Settings

When you are working behind a proxy you need to configure it for your virtualized environments. This is necessary, because *Chef* and *Yum* need to access artifacts from remote locations such as RPMs in case of *Yum* and Gems in case of *Chef*.

Proxy Setup with Vagrant and Chef

To set up the proxy for the virtualized *Vagrant* environment, you need to configure the proxy URL and optionally its authentication credentials in the *Vagrantfile* or in one of the override locations provided by the *nugrant* plugin.

To configure the proxy settings in the *Vagrantfile*, you need to adapt the proxy properties of the `config.user.defaults` hash. To configure a proxy externally, that is on a user-based level, you need to add a `~/.vagrantuser` file and add the following content:

```
proxy:
  url: "http://myproxy"
  username: "my username"
  password: "mypassword"
```

Proxy Setup with Chef only

When you trigger the provisioning with *Chef*, either using *chef-solo* or a *chef-client* together with *Chef Server*, you need to configure your proxy settings separately for *Chef* and *Yum*.

For the *Yum* proxy settings you need to configure the `proxy` attributes of the *yum* cookbook. The documentation of the cookbook can be found in the `boxes/chef/chef-repo/cookbooks/yum/README.md`. You can configure the

proxy settings either in a `recipe`, a `role` or as recommended in an `environment`. In case you are using `chef-solo` you can use the `node.json` file.

To configure the proxy for *Chef* itself, you need to configure the proxy properties in the corresponding *Chef* configuration files. For `chef-solo`, this would be the `solo.rb` file and for a `chef-client` the `client.rb`. The properties you need to configure are:

- ➔ `http_proxy`
- ➔ `https_proxy`
- ➔ `http_proxy_user`
- ➔ `http_proxy_pass`

For a complete reference, visit the official [Chef Documentation](#).

3.3.5 Configuring Local Setup

Database Setup

If you don't want to use the virtualized setup, you need to create different databases and users used by the various *CoreMedia CMS* components (see [Section 3.2, “Pre-requisites”](#) [39]). At the root level of the *Blueprint* workspace you will find SQL scripts for creating and dropping all database entities needed for the relational database. A default MongoDB installation requires no further configuration, because the *CoreMedia* components connect through the default port with no user credentials, by default. If required, see [Section 4.5, “Collaborative Components”](#) in *CoreMedia Operations Basics* for more MongoDB configuration.

The scripts are suitable for a local MySQL instance in a developer environment. You can easily adapt them for other databases or remote users. There are also Bash scripts and Windows batch files to apply the SQL scripts. If the MySQL server is running and the `mysql` command line client is executable via the `PATH` variable, you only need to execute the following in order to prepare the databases for *CoreMedia DXP 8*.

Windows:

```
> cd $CM_BLUEPRINT_HOME\workspace-configuration\database\mysql\
createdB.bat
```

Linux:

```
$ cd $CM_BLUEPRINT_HOME/workspace-configuration/database/mysql
./createDB.sh
```

The command was successful if the following databases have been created:

Table 3.3. Database Settings

Database	User	Password	Description
cm7manage- ment	cm7manage- ment	cm7manage- ment	Database for the <i>Content Manage- ment Server</i>
cm7master	cm7master	cm7master	Database for the <i>Master Live Server</i>
cm7replication	cm7replication	cm7replication	Database for the <i>Replication Live Server</i>
cm7mcafeeder	cm7mcafeeder	cm7mcafeeder	Database for the <i>CAE Feeder</i> connec- ted to the <i>Content Management Server</i>
cm7cafeeder	cm7cafeeder	cm7cafeeder	Database for the <i>CAE Feeder</i> connec- ted to the <i>Master Live Server</i>

Host Mappings

In order to run *Blueprint* locally, you have to add the following mappings to the `etc/hosts` file:

```
127.0.0.1      studio-helios.localhost
127.0.0.1      preview-helios.localhost
127.0.0.1      helios.localhost
127.0.0.1      studio.corporate.localhost
127.0.0.1      preview.corporate.localhost
127.0.0.1      corporate.localhost
127.0.0.1      editor.localhost
127.0.0.1      shop-preview-helios.localhost
127.0.0.1      shop-helios.localhost
127.0.0.1      shop-preview-production-helios.localhost
```

3.3.6 In-Memory Replacement for MongoDB-Based Services

Several *CoreMedia* core features like *CapLists*, *notifications* and *projects/to-dos* use *MongoDB* as a persistence layer. Although not recommended it is possible to substitute *MongoDB* with an *in-memory* persistence layer.

There is no *in-memory* replacement for the persistence layer of the *Elastic Social* extension. *MongoDB* is required for that.

Besides not supporting *Elastic Social* there are other functional limitations to the *in-memory* approach. Collaboration based on *projects/to-dos* will not work properly with more than one *Studio* server.





In order to activate the in-memory persistence for *Studio* and the *Workflow server*, update Blueprint extensions, using the Maven profile `controlroom-memory`. It will add all necessary Maven dependencies to *Studio* and *Workflow Server*, in order to persist Control Room data in-memory. For more information how to update Blueprint extensions, see [Section “Adding, Disabling or Removing an Extension” \[147\]](#)

```
mvn com.coremedia.maven:\
coremedia-blueprint-maven-plugin:\
update-extensions -Pcontrolroom-memory
```

In-Memory configuration for the Studio

You need to configure the *Studio* not only to use the *in-memory* persistence layer but also to be the only *User Changes* web application.

The following Maven dependencies have to be available in the `studio-webapp` module.

```
<dependency>
  <groupId>com.coremedia.ui</groupId>
  <artifactId>collaboration-memory-rest-component</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>${project.groupId}</groupId>
  <artifactId>controlroom-memory-plugin</artifactId>
  <version>${project.version}</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>cap-client-list-memory</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>notification-elastic</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>project-elastic</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.ui.collaboration</groupId>
  <artifactId>user-changes-component</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.ui.collaboration</groupId>
  <artifactId>workflow-notifications-component</artifactId>
  <scope>runtime</scope>
</dependency>
```

Add the following property to *Studio*'s `application.properties`.

```
models.createIndexes=false
```

Add the following customizer to the `beans` section of the `application.xml` of the `studio-webapp` module.

```
<customize:append id="uapiMemoryCapListConnector"
bean="connectionParameters">
  <map>
    <entry key="usecaplist" value="true"/>
    <entry key="caplist"
value="com.coremedia.cotopaxi.list.memory.MemoryCapListConnectorFactory"/>
  </map>
</customize:append>
```

Also, configure your *Studio* server with the following properties, so that the *in-memory* store is read / written from the given file upon application context startup / shutdown of *Studio*. To limit memory usage of the in-memory store, the size per collection map is configured. To be robust against data loss, the in-memory store can be persisted periodically in a given interval.

Property	Default	Description
memory.collection.serialization.file	null	In-memory store persistence file name.
memory.collection.size	5000	Number of in-memory map entries per collection.
memory.collection.serialization.interval	360000	Interval in ms in which the in-memory store is persisted periodically to the configured file. If 0, periodic persistence is disabled.
memory.collection.selfclearing.names	notifications	A comma separated list of collection names, which will be periodically deleted and re-created, when <code>memory.collection.size</code> is reached. Fast growing collections, which do not contain critical data should be configured as self-clearing collections, e.g. notifications.

Table 3.4. *Studio Configuration Properties for In-Memory Store*

In-Memory configuration for the Workflow Server

In the *in-memory* deployment after starting and finishing workflows, the *Workflow Server* sends the collected data for pending and finished processes to the *Studio* server, where it is persisted in *Studio*'s *in-memory* persistence layer. In order to connect to *Studio*, the *Workflow Server* needs an authorized user. Therefore, configure the *Workflow Server* to use the following properties. Additionally, disable `CapLists` for the *Workflow Server*.


```
studio.url=http://<host>:<port>/<context>  
studio.user=<username>  
studio.password=<userpassword>  
  
workflow.usecaplist=false
```

You also have to exclude all *Elastic* dependencies from the *Workflow Server*, in order to prevent that *MongoDB* is used and not available. The following dependency has to be available in the *Workflow Server* web application.

```
<dependency>  
  <groupId>com.coremedia.cms</groupId>  
  <artifactId>cap-workflow-archive-memory</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

3.4 Customizing IBM WebSphere Commerce

Only required when you want to use the e-Commerce Blueprint



This section describes how you have to adapt your IBM Rational Application Developer (RAD) environment in order to integrate with *CoreMedia Digital Experience Platform 8*.

In general, certain configuration files need to be adapted in the IBM WebSphere Commerce workspace. Depending on your degree of already applied customization, you might need to merge the provided configuration snippets with your custom code.

This chapter also contains small configurations in the CoreMedia system. These tasks are highlighted in the margin.

Deployment to IBM WebSphere Commerce servers, including Staging, Production and Development, is not part of this manual. Please refer to appropriate IBM documentation in the info center at <http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/index.jsp>



The configuration should be performed by an experienced RAD developer.

Scope of delivery

In order to connect *CoreMedia DXP 8* with your IBM WebSphere Commerce server you will get the following artifacts from CoreMedia:

- The *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive (Workspace archive, for short). It contains the required resources to customize the IBM WebSphere Commerce Server and JAR files with extensions for *CoreMedia DXP 8* to be added to the classpath of your IBM WebSphere Commerce workspace and deployment packages. These files include the configuration described in the following chapters.
- The *CoreMedia LiveContext 2.0 WebSphere Commerce Project Sample Data* archive (Sample Data archive, for short). The archive contains sample data for the WCS system, which corresponds with the test data for the CoreMedia system in *CoreMedia Blueprint*.

You will find both files on the CoreMedia releases download page at <http://releases.coremedia.com/dxp8>

The customization involves the following aspects:

Installation steps

1. [Section 3.4.1, “Preparing the RAD Workspace” \[60\]](#) describes how to apply the required customization to your IBM WebSphere Commerce workspace
2. [Section 3.4.2, “Copy Libraries” \[60\]](#) describes how to copy required libraries into the WCS.
3. [Section 3.4.3, “Configuring the Search” \[60\]](#) describes how you have to extend the IBM search profile and the Solr index. This enables the CoreMedia system to get additional information necessary for the integration.
4. [Section 3.4.4, “Extending REST Resources to BOD Mapping” \[65\]](#) describes how you have to configure the mapping of REST resources to the Business Object Document nouns.
5. [Section 3.4.5, “Configuring the Cookie Domain” \[65\]](#) describes how you enable session synchronization between the CoreMedia and IBM system for content-led scenario.
6. [Section 3.4.6, “Multiple Logon for the Same User” \[66\]](#) describes how you configure the IBM WCS system to accept multiple logins with the same user.
7. [Section 3.4.7, “Configuring REST Handlers” \[67\]](#) describes which REST handlers you have to add and configure.
8. [Section 3.4.8, “Applying Changes to the Management Center” \[68\]](#) describes the deployment of the Management Center customization.
9. [Section 3.4.9, “Deploying the CoreMedia Fragment Connector” \[68\]](#) describes the deployment of the fragment connector, which renders content from *CoreMedia DXP 8* as fragments to IBM WebSphere Commerce pages.
10. [Section 3.4.10, “Customizing IBM WebSphere Commerce JSPs” \[72\]](#) describes how to apply customizations to IBM WebSphere Commerce JSPs.
11. [Section 3.4.11, “Deploying the CoreMedia Widgets” \[73\]](#) describes the deployment of the CoreMedia widgets, which can be used to add content or assets from *CoreMedia DXP 8* to IBM WebSphere Commerce pages using the fragment connector.
12. [Section 3.4.12, “Setting up SEO URLs for CoreMedia Pages” \[77\]](#) describes how to set up SEO URLs for CoreMedia Pages.
13. [Section 3.4.13, “Event-based Commerce Cache Invalidation” \[78\]](#) describes how to enable event based commerce cache invalidation.
14. [Section 3.4.14, “Deploying the CoreMedia Catalog Data” \[79\]](#) describes how to import the CoreMedia catalog content from the Sample archive into the WCS.

In the following sections `WCDE-INSTALL` stands for the installation directory of your IBM WebSphere Commerce RAD installation.



3.4.1 Preparing the RAD Workspace

CoreMedia Digital Experience Platform 8 integrates with IBM WCS using the WebSphere Commerce REST API, therefore you have to deploy/enable all the REST modules in the WCS workspace for *CoreMedia DXP 8* to function properly. These modules include: `Rest` and `Search-Rest` modules.

REST modules

The *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive (download at <http://releases.coremedia.com/dxp8>) contains all new and extended files required to install *CoreMedia DXP 8* in the IBM WebSphere Commerce RAD workspace. In principle, you can copy the workspace on top of a fresh Aurora RAD workspace, but only when you do not already have customizations. Make sure you download the Zip archive that matches your WebSphere Commerce version.

Content of the ZIP file

If you have already customized the Aurora RAD workspace, you cannot copy the CoreMedia Zip content above it, because this would overwrite the former changes. In this case, unzip the file and add and merge the files manually as described in the subsequent sections.



3.4.2 Copy Libraries

Copy the content of the `workspace/WC/lib/` folder of the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive file into the IBM RAD workspace folder `workspace/WC/lib/`

Make sure that the following files from the CoreMedia workspace archive are in the corresponding locations of the WCS workspace:

- `workspace/Stores/WebContent/WEB-INF/lib/coremedia-livecontext-wcs-<version>.jar`
- `workspace/Rest/WebContent/WEB-INF/lib/coremedia-livecontext-wcs-<version>.jar`
- `workspace/Search-Rest/WebContent/WEB-INF/lib/coremedia-livecontext-wcs-<version>.jar`

3.4.3 Configuring the Search

WebSphere Commerce search provides enhanced search functionality to a store and also influences the search results by using search term association and search-based merchandising rules. In this section you will adapt WebSphere Commerce search to allow *CoreMedia DXP 8* to leverage these search features. This includes browsing and searching of all catalog assets in *CoreMedia Studio* which is the editorial interface of *CoreMedia DXP 8*. The configuration consists of two tasks:

1. Extend the search profiles
2. Add a new field to the Solr index

Extending Search Profiles

In WebSphere Commerce Search, search profiles (defined in the `wc-search.xml` configuration file) are used to control the storefront search experience at a page level by grouping sets of search runtime parameters. The search runtime parameters set needs to be extended to support the feature set introduced by *CoreMedia DXP 8*.

CoreMedia DXP 8 requires additional information like SEO identifier or pricing which the WebSphere Commerce REST API does not provide by default. Providing this information via REST API is achieved by extending the `wc-search.xml` configuration file to include the new search profiles definition that extends the existing profiles.

*Additional information
for LiveContext*

To change/add the value of an existing property in the WebSphere Commerce search configuration file, you have to create a customized version of this file containing only the changed properties. Follow the steps below to extend the search profiles:

1. Add search profiles:

Open the file `WCDE-INSTALL/workspace/Search/xml/config/com.ibm.commerce.catalog-ext/wc-search.xml` in the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* and copy all the `config:profile` definitions with a name starting with *CoreMedia* to the corresponding file in your WCS RAD workspace.

2. You have to extend the existing REST API search handlers to provide the additional information now exposed by the search profiles.

Change the search profile for existing search based REST handlers by creating/updating the file `WCDE-INSTALL/workspace/Search-Rest/WebContent/WEB-INF/config/com.ibm.commerce.rest-ext/wc-rest-resourceconfig.xml` with the corresponding changes from the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive.

Enabling Dynamic Pricing

Dynamic Pricing supports different prices for different B2B contracts. By default, the feature is disabled.

You activate dynamic pricing by an update of the `STORECONF` table. Set the `wc.search.priceMode` property in the `STORECONF` table to value "2". See also http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchstoreconf.htm?lang=en

Customizing the IBM WCS Solr Index

CoreMedia DXP 8 comes with Solr schema customizations to be applied to the IBM WCS Solr schema definition.

The schema customization can be found in the WCS RAD workspace zip file below `WCDE-ZIP/components/foundation/subcomponents/search/solr/cm-schema-customizations/CatalogEntry/schema.xml` and `WCDE-ZIP/components/foundation/subcomponents/search/solr/cm-schema-customizations/CatalogGroup/schema.xml`.

Add the additional fields and fieldTypes to the corresponding `schema.xml` files below `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template\` and `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template-update\` (if existing) to your *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace*.

The file `WCDE-ZIP/components/foundation/subcomponents/search/solr/cm-schema-customizations/merge.xml` can be used to add the *CoreMedia* Solr customization to your existing Solr schema files in your WCS deployment automation.

Read [Section “Adding New PARENT_PARTNUMBER Field to the Solr Index” \[62\]](#) and [Section “Adding New CM_SEO_TOKEN Field to the Solr Index ” \[63\]](#) to learn more about the specific fields in detail.

Adding New PARENT_PARTNUMBER Field to the Solr Index

Searching IBM WCS catalog assets in *CoreMedia Studio* is part of the seamless integration experience that *CoreMedia DXP 8* brings to the table. Almost all the catalog assets are searchable in *CoreMedia DXP 8* without any need of customization except for the catalog product asset which acts as a template for a group of items (or SKUs) that exhibit the same attributes.

This needs an extra property to explicitly define the hierarchical relationship between the product and its variants in order to make the variants also searchable in *Studio*. This subsection describes all the steps required to introduce the custom *CoreMedia Digital Experience Platform 8* parent part number field which establishes the relationship between product and variant in WebSphere Commerce.

1. Preprocessing data involves querying WebSphere commerce tables and creating a set of temporary tables to hold the data. The file `WCDE-INSTALL\components\foundation\samples\dataimport\catalog\oracle\wc-dataimport-preprocess-parent-partnumber.xml` in the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* defines a custom preprocessing task for this. The file contains the new temporary table definition, database schema metadata, and a reference to the Java class used in the preprocessing steps for an Oracle database.

Simply copy the file to the corresponding location in your IBM WCS RAD system. The workspace contains files for other databases which you can use similarly.

2. Add the following new field to the Solr schema in the file `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\locale\en_US\schema.xml` and in the corresponding files for other languages:

```
<field name="parent_partNumber ntk"
      type="wc_keywordTextLowerCase" indexed="true"
      stored="true" multiValued="false"/>
```

Example 3.2. New Solr field

3. Now, you have to configure data extraction from the relational table. The Data extraction is handled by the data import handler, containing configuration files with predefined SQL query lines that extract WebSphere Commerce data. You have to extend the extraction scope by including parent part number into the SQL statements.

In the file `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\database\oracle\wc-data-config.xml` (and corresponding files for other databases) you have to adapt some lines. To do so, proceed as follows:

- a. Open the file `components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\database\oracle\wc-data-config.xml` in the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace*.
 - b. In the file, search for all entries that contain the string `TI_PARENTCHILDCATALOGENTRY`. Copy these lines to the corresponding positions in the `wc-data-config.xml` file of your WCS RAD workspace.
 - c. In the file, search for comments that contain the string `CoreMedia` and copy the lines surrounded by these comments to the corresponding positions in the `wc-data-config.xml` file of your WCS RAD workspace.
4. Rebuild the index as described in the IBM documentation at <http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdmanagesearchpopbuild.htm>

WebSphere Commerce search contains a scheduler job (`UpdateSearchIndex`) to synchronize the catalog changes with the search index. The default update interval is 5 minutes. You can change this default value according to your needs in the WebSphere Commerce Administration Console.

Adding New CM_SEO_TOKEN Field to the Solr Index

Per default IBM behaviour, you cannot distinguish the SEO keyword overridden by a store. If you have overridden the SEO keyword in the store, then you will get

multiple SEO keywords in the response, without knowing which SEO keyword belongs to which store. To be able to distinguish the SEO keyword you need to extend the Solr field by adding the custom CM_SEO_TOKEN field in the Solr index. This custom CM_SEO_TOKEN field concatenates the store ID and the SEO keyword.

1. Add a preprocessing file for CM_SEO_TOKEN field. The file `WCDE-INSTALL\components\foundation\samples\dataimport\catalog\oracle\wc-dataimport-preprocess-cm-seo-token.xml` in the CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace defines a custom preprocessing task for this. The file contains the new temporary table definition, database schema metadata and a reference to the Java class used in the preprocessing steps for an Oracle database.

Copy the file to the corresponding location in your IBM WCS RAD system. The workspace contains files for other databases which you can use similarly.

2. Add a new field to the Solr schema for the CatalogEntry and for the CatalogGroup as shown in `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\locale\en_US\schema.xml` and in the corresponding files for other languages:

```
<field name="cm_seo_token_ntk" type="wc_cmKeywordTextLowerCase"
indexed="true" stored="true" multiValued="false" />
```

*Example 3.3. New
CM_SEO_TOKEN Solr
field*

3. Configure data extraction from the relational table. Data extraction is handled by the data import handler, containing configuration files with predefined SQL query lines that extract WebSphere Commerce data. You have to extend the extraction scope by including CM_SEO_TOKEN into the SQL statements.

In the file `WCDE-INSTALL\components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\database\oracle\wc-data-config.xml` (and corresponding files for other databases) you have to adapt some lines. To do so, proceed as follows:

- a. Open the file `components\foundation\subcomponents\search\solr\home\template\CatalogEntry\conf\database\oracle\wc-data-config.xml` in the CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace.
- b. In the file, search for all entries that contain the string `TI_CM_SEOURL`. Copy these lines to the corresponding positions in the `wc-data-config.xml` file of your WCS RAD workspace.
- c. In the file, search for comments that contain the string `CoreMedia` and copy the lines surrounded by these comments to the corresponding positions in the `wc-data-config.xml` file of your WCS RAD workspace.

4. Rebuild the index as described in the IBM documentation at <http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdmanagesearchpopbuild.htm>

WebSphere Commerce search contains a scheduler job (UpdateSearchIndex) that synchronizes catalog changes with the search index. The default update interval is 5 minutes. You can change the default value in the WebSphere Commerce Administration Console.

3.4.4 Extending REST Resources to BOD Mapping

The BOD Mapping only needs to be extended if you do not make use of the search based REST handlers. Per default search based REST handlers are active and there is no need to apply the following.



In order to retrieve more detailed information from the REST handlers, the mapping of the REST resources to the Business Object Document (BOD) nouns has to be extended.

1. To retrieve the SEO identifier of a product, create and edit the file `WCDE-INSTALL/workspace/Rest/WebContent/WEB-INF/config/bodMapping-ext/rest-productview-clientobjects.xml` accordingly to the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive.
2. To retrieve the SEO identifier of a category, create and edit the file `WCDE-INSTALL/workspace/Rest/WebContent/WEB-INF/config/bodMapping-ext/rest-categoryview-clientobjects.xml` accordingly to the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive.

3.4.5 Configuring the Cookie Domain

This is only necessary in the content-led scenario or when you use AJAX calls (for Elastic Social, for example) in the commerce-led scenario.



Since the CAE must know about generated commerce cookies and vice versa, it is necessary to configure specific cookie domains for both, the CAE and the commerce system, so that cookies are exposed to both systems.

The CoreMedia system must be hosted on servers belonging to the same domain. Example: If the domain given here is `.xyz.com`, then the CoreMedia CAE must be accessed from the WCS via a (logical) server name `servername.xyz.com`.

In order to enable session synchronization between the CoreMedia and IBM system do the following steps:

1. Enable the `com.coremedia.livecontext.hybrid.CookieLeveler` within the `web.xml` of your commerce store front (`WCDE-INSTALL/workspace/Stores/WebContent/WEB-INF/web.xml`) and preview (`WCDE-INSTALL/workspace/Preview/WebContent/WEB-INF/web.xml`) webapp. Put its filter mapping in front of all other filter mappings and set the cookie domain to the shared domain of your CAE and WCS.

```
<filter>
  <filter-name>Cookie Leveler</filter-name>

  <filter-class>com.coremedia.livecontext.hybrid.CookieLeveler</filter-class>
</filter>
<filter-mapping>
  <filter-name>Cookie Leveler</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<context-param>

<param-name>com.coremedia.fragmentConnector.cookieDomain</param-name>
  <param-value>.blueprint-box.vagrant</param-value>
</context-param>
```

2. If you want to rewrite additional cookies of your own commerce customizations, you can configure a list of all cookies to be rewritten via a comma separated list.

```
<filter>
  <filter-name>Cookie Leveler</filter-name>


  <filter-class>com.coremedia.livecontext.hybrid.CookieLeveler</filter-class>

  <init-param>
    <param-name>cookieFilter</param-name>
    <param-value>WC_,WCP_,myCustomCookie</param-value>
  </init-param>
</filter>
```

Per default commerce cookies starting `WC_` and `WCP_` are mapped.

3. Set the cookie domain for the `JSESSION` cookie on the commerce server, for example via the IBM console.

For the CAE the cookie domain is configurable in the Tomcat configuration file `context.xml`. The cookie domain can be set here by setting the attribute `sessionCookieDomain` for the `Context` element. The cookie domain must be configured with a leading "." so that the CAE session cookie is readable from commerce system that run with the same subdomain, for example `.myDomain.com`.

 *Configure in the CoreMedia system*

3.4.6 Multiple Logon for the Same User

Since all CAE and Studio instances must authenticate against *IBM WebSphere Commerce Server* for protected REST requests, and by default all instances use the same technical username, it is required to configure your *IBM WebSphere Commerce*

Server to allow multiple logins per user at the same time using the `AllowMultipleLogonForSameUser` property.

Otherwise, multiple clients (CAE and Studio, for instance) will terminate each others session and need to re-login frequently, causing long delays on the REST communication layer.

Please refer to [IBM Knowledge Center](#) for details on how to set the `AllowMultipleLogonForSameUser` property.

3.4.7 Configuring REST Handlers

CoreMedia DXP 8 requires additional REST handlers and some configuration of existing handlers.

Adding New REST Handlers

CoreMedia LiveContext API comes with additional REST handlers in order to make more data accessible and to provide additional data processing capabilities. The handler classes reside in the `WebSphereCommerceServerExtensionsLogic` module.

You have to add the following handlers:

LanguageMapHandler	The <code>LanguageMapHandler</code> returns a list of all available languages of the WebSphere Commerce Server with its mapping on the internal language identifier which is used for certain REST calls.
StoreInfoHandler	The <code>StoreInfoHandler</code> returns the stored and the catalog information of all available stores in the WebSphere Commerce Server.
CacheInvalidationHandler	The <code>CacheInvalidationHandler</code> returns invalidation events from the <code>CACHEIVL</code> table (see also Section 3.4.13, “Event-based Commerce Cache Invalidation” [78]).
WorkspacesHandler	The <code>WorkspacesHandler</code> is used to display available commerce workspaces in studio.

The following handler is only necessary for content-led integration.

ResetPasswordHandler	The <code>ResetPasswordHandler</code> is used to update the users passwords. This handler is only neces-
-----------------------------	--



sary for content-led integration with *IBM WCS (FEP7)*

In order to add the handlers proceed as follows:

1. Add the CoreMedia LiveContext library package to the `Rest` module in your commerce development workspace.
2. Add the following fully qualified names of the handlers to the file `WCDE-INSTALL/workspace/Rest/WebContent/WEB-INF/config/resources-ext.properties` accordingly to the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive.
3. Add a resource element for each handler to the file `WCDE-INSTALL/workspace/Rest/WebContent/WEB-INF/config/com.ibm.commerce.rest-ext/wc-rest-resourceconfig.xml` accordingly to the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive.
4. For the `CacheInvalidationHandler` add the file `WCDE-INSTALL/workspace/WC/xml/config/com.ibm.commerce.catalog-ext/wc-query-CoreMedia-LiveContext.tpl` from the the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive. The file contains a database template to access *IBM WCS CACHEIVL* table.
5. Adapt all `dbtype` properties to your target database.

3.4.8 Applying Changes to the Management Center

Studio integrates the Management Center into its GUI. For the integration do as follows:

1. Add the file `WCDE-INSTALL/workspace/LOBTools/WebContent/CoreMediaManagementCenterWrapper.html` from the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive to the `LOBTools` module.

This file is used from *CoreMedia Studio* for displaying products, categories and e-Marketing Spots in the *IBM WebSphere Commerce Management Center*. The wrapper uses the original *IBM Management Center* JSP files embedded and delegates deep links to the appropriate *IBM* functions.

3.4.9 Deploying the CoreMedia Fragment Connector

The *CoreMedia Fragment Connector* is the component that connects with *CoreMedia CAE* in order to integrate *CoreMedia* content fragments in store pages. In order to perform a fragment request, the `LiveContextEnvironment` has to be configured in the `WCDE_installdir/workspace/Stores/WebContent/WEBINF/web.xml` configuration file, as described below.

Changing the `web.xml` file

There are different approaches to configure the loading mechanism for properties for the fragment connector. The `LiveContextEnvironment` can load its configuration directly from `web.xml`, from a properties file and from the `STORECONF` table. The default implementation is `PropertiesBasedIBMLiveContextEnvironmentFactory`.

The `PropertiesBasedIBMLiveContextEnvironmentFactory` extends the `IBMLiveContextEnvironmentFactory` and in addition loads properties from a resource file on the classpath. If the resource file cannot be found - or the resource cannot be loaded, it will throw `RuntimeExceptions`. The location of the properties resource must be given in a servlet context parameter named `livecontext.properties.location`. In the first place this factory tries to get a parameter from `STORECONF` table, in the second place from the properties file and if not found as fallback from `web.xml`.

Other approaches are the following:

- ➔ The `DefaultLiveContextEnvironmentFactory` reads the connector properties directly as context parameters directly from the `web.xml`.
- ➔ The `IBMLiveContextEnvironmentFactory` extends the `DefaultLiveContextEnvironmentFactory` and can be configured via the `STORECONF` table. If properties are not available in the `STORECONF` table the factory reads directly from the `web.xml` configuration.

The fragment connector is the central component in the commerce-led integration scenario (see [Section 5.1, “Commerce-led Integration Scenario” \[176\]](#)). Configure the fragment connector for example as follows:

1. Add the `LiveContextEnvironment` configuration as shown in `WCDE-INSTALL/workspace/Stores/WebContent/WEB-INF/web.xml` to the corresponding file in the WCS RAD workspace.
2. In the file `WCDE-INSTALL/workspace/Stores/WebContent/WEB-INF/coremedia-connector.properties` configure at least the parameter `com.coremedia.fragmentConnector.liveCaeHost` with the host URL of your Content Application Engine (CAE). If you use a single WCS that should be able to connect to both, preview and production CAE, you also need to set `com.coremedia.fragmentConnector.previewCaeHost` with the host URL of the preview CAE. In case you have a dedicated Staging WCS with separate Production System, you only need to configure one CAE host, each. Find the meaning of all parameters in the list below.

`com.coremedia.fragmentConnector.cookieDomain`

The `cookieDomain` is used when a fragment request is created. All accessible cookies are copied and added to this re-

`com.coremedia.fragmentConnector.environment`

quest using the specified cookie domain. This way it is ensured that the CAE session cookie is detected by the CAE and fragments can be rendered depending on the logged on user. The `cookieDomain` can contain multiple `cookieDomains` separated by comma.

The optional parameter is used to identify the WCS that is requesting a fragment from a CAE. It may be used to serve different sites for each WCS that is connected to a single CMS. The strategy for resolving this parameter is implemented in the class `LiveContextSiteResolver`. The method `findSiteFor(@NonNull FragmentParameters fragmentParameters)` checks if the `environment` parameters has been passed as request matrix parameter. If set (for example: `site:PerfectChef`), a lookup is made if a site with a matching name and locale exists. If no site is found with the given name, the default lookup strategy, implemented in `findSiteFor(@NonNull String storeId, @NonNull Locale locale)` is used.

`com.coremedia.fragmentConnector.liveCaeHost`

The `liveCaeHost` identifies the Live CAE, to be precise, the Varnish, Apache or any other proxy in front of the Live CAE. Each request made by the fragment connector will be prefixed with the `urlPrefix`.

`com.coremedia.fragmentConnector.previewCaeHost`

The `previewCaeHost` identifies the Preview CAE, to be precise, the Varnish, Apache or any other proxy in front of the Preview CAE. Each request made by the fragment connector will be prefixed with the `urlPrefix`. The `previewCaeHost` is only required if you want a single WCS instance being able to access the preview CAE in case of WCS preview and the live CAE in all other cases. Additionally, the preview mode can be invoked through an HTTP header as described in [Section 3.5.4, "Developing with Apache \(optional for e-Commerce\)" \[91\]](#). If you have a dedicated WCS instance for staging

`com.coremedia.fragmentConnector.urlPrefix`

and separate production WCS, you do not need to set this property. If this parameter is not set, the parameter `liveCaeHost` will be used instead.

This prefix identifies the web application, the servlet context and the fragment handler to handle fragment requests. The default request mapping of all the handlers within *CoreMedia Blueprint* that are able to handle fragment requests start with `service/fragment`.

`com.coremedia.widget.templates`

Configures the template lookup path that is used when rendering CoreMedia Widget includes. Default is `/Widgets-CoreMedia/com.coremedia.commerce.store.widgets.CoreMediaContentWidget/impl/templates/`

`com.coremedia.fragmentConnector.defaultLocale`

Every fragment request needs to contain the tuple `(storeId, locale)` because it is needed to map a request to the correct site. Using `defaultLocale` you can set a default that is used for every request that does not contain a custom locale. You will see how it is used later, when you see the `IncludeTag` in action.

`com.coremedia.fragmentConnector.contextProvidersCSV`

Every fragment request can be enriched with shop context specific data. It will be most likely user session related info, that is available in the *IBM WCS* and can be provided to the back-end CAE via a `ContextProvider` implementation. See [Section 5.1.3, “Extending the Shop Context in Commerce-led Integration Scenario” \[181\]](#) for details.

`com.coremedia.fragmentConnector.isDevelopment`

The fragment connector will return error messages that occur in the CAE while rendering a fragment if the `isDevelopment` parameter is set to true. For production environments you should set this option to `false`. Errors are logged then but do not appear on the commerce page so that the end user will not recognize the errors.

`com.coremedia.fragmentConnector.disabled`

Turn this flag to true if you want to disable the fragment connector. Disabled means that the fragment connector always delivers an empty fragment. This property is not mandatory. If this property is not set the default is false.

`com.coremedia.fragmentConnector.connectionTimeout`

The connection timeout in milliseconds used by the fragment connector; that is the time to establish a connection. A value of "0" means "infinite". Default is "10000".

`com.coremedia.fragmentConnector.socketTimeout`

The socket read timeout in milliseconds used by the fragment connector; that is the time to wait for a response after a connection has successfully been established. A value of "0" means "infinite". Default is "30000".

`com.coremedia.fragmentConnector.connectionPoolSize`

Maximum number of connections used by the fragment connector. Default is 200.

3.4.10 Customizing IBM WebSphere Commerce JSPs

When the CoreMedia Fragment Connector has been installed, the `lc:include` tag can be used in any JSPs of the Commerce Workspace to include content from the CoreMedia CMS. See [Section “The CoreMedia Include Tag” \[186\]](#) for more details.

The *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* contains web content like JSP and JavaScript files in the `Stores/<STORE_NAME>` folder. These files are mostly adapted versions of the JSP files of an original IBM RAD workspace. The CoreMedia customizations are highlighted with the following comment lines:

```
<!-- Begin CoreMedia XXX -->
CoreMedia snippet data
<!-- END CoreMedia XXX -->
```

The corresponding files in the IBM RAD workspace are in the `workspace/Stores/WebContent/<STORE_NAME>` folder.

If you have an Aurora RAD workspace without any customizations, you can copy the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive content above it. Otherwise, you have to unzip the file and check for each file if you can copy the CoreMedia change into the corresponding file of your IBM RAD workspace.

How to adapt the files

Example

The CoreMedia archive contains custom `Header.jsp` and `Footer.jsp` files. These JSPs contain some `include` tags, highlighted with comments, to replace the default Aurora store header and footer with CoreMedia page grid placements. The placements contain the navigation and footer elements of the CAE. The original files are located in the folder `workspace/Stores/WebContent/<STORE_NAME>/Widgets` of the RAD workspace.

In addition CoreMedia JavaScript and CSS that is used by the CAE must be included in the store front. To do so adapt the CoreMedia specific changes in `WebContent/<STORE_NAME>/Common/CommonJSToInclude.jspf`.

3.4.11 Deploying the CoreMedia Widgets

The CoreMedia widgets are IBM Commerce Composer Widgets. You can use the *CoreMedia Content Widget* to add CoreMedia content fragments to your IBM WebSphere pages and the *CoreMedia Asset Widget* to add product images to product detail pages.

The Asset Widget is part of *CoreMedia Advanced Asset Management* which requires separate licensing.



Prerequisites

In order to use the CoreMedia widgets to embed CoreMedia fragments, the Fragment Connector needs to be deployed before executing these steps.

Register the Widget definition and subscribe your Store to it

See the IBM documentation at https://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.data.doc/concepts/cmlbatchoverview.htm for more details about data load.

1. Stop the IBM WebSphere Commerce server in the IBM RAD environment.
2. Adapt the database settings in the *Data Load* environment configuration file (`SAMPLEDATA-ZIP\workspace\DataLoad\dataload\common\wc-data-load-env.xml`) from the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Sample Data* Zip file to the settings of your WebSphere database.

You can retrieve your database settings from the IBM RAD environment configuration file `WC/xml/config/wc-server.xml`, at the following XML element:

```
<InstanceProperties>
  <Database>
    <DB>
```

For a DB2 database, the attribute `schema` in `wc-dataload-env.xml` corresponds to the attribute `DBNode` in `wc-server.xml`.

Find your store identifier in the IBM Management Center in *Store Management*. If you use the default IBM shop, the value is "Aurora".

3. Use the *Data Load* business object configuration files from the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Sample Data* ZIP file for registering the widget definition (`workspace\DataLoad\dataload\common\[store_name]\Widget\wc-loader-registerWidgetdef.xml`) and for subscribing the widget definition (`workspace\DataLoad\dataload\common\[store_name]\Widget\wc-loader-subscribeWidgetdef.xml`) where `store_name` is the store identifier of your store ("AuroraESite", for instance).
4. Use the CSV input files from the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Sample Data* ZIP file for registering the widget definition (`workspace\DataLoad\dataload\common\[store_name]\Widget\registerWidgetdef.csv`) and for subscribing the widget definition (`workspace\DataLoad\dataload\common\[store_name]\Widget\subscribeWidgetdef.csv`).
5. Configure the *Data Load* order configuration file (`wc-dataload.xml`). The *Data Load* file has pointers to the environment settings file, the business object configuration file and the input file.

```
<?xml version="1.0" encoding="UTF-8"?>

<_config:DataLoadConfiguration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.ibm.com/xmlns/prod/commerce/foundation/config
    ../../../../xml/config/xsd/wc-dataload.xsd"
  xmlns:_config=
    "http://www.ibm.com/xmlns/prod/commerce/foundation/config">

  <_config:DataLoadEnvironment configFile="wc-dataload-env.xml"/>

  <_config:LoadOrder commitCount="100"
    batchSize="1"
    dataLoadMode="Replace">
    <_config:property name="firstTwoLinesAreHeader" value="true"/>
    <_config:property name="loadSEO" value="true"/>

    <!-- Configuration for the file to register a widget -->
    <_config:LoadItem
      name="RegisterWidgetDef"
      businessObjectConfigFile=
        "wc-loader-registerWidgetdef.xml">
      <_config:DataSourceLocation
        location="registerWidgetdef.csv"/>
    </_config:LoadItem>

    <!-- Configuration for the file to subscribe a store to a
```

Example 3.4. *wc-dataload.xml*

```
widget -->
  <_config:LoadItem
    name="SubscribeWidgetDef"
    businessObjectConfigFile=
      "wc-loader-subscribeWidgetdef.xml">
    <_config:DataSourceLocation
      location="subscribeWidgetdef.csv"/>
  </_config:LoadItem>
</_config:LoadOrder>
</_config:DataLoadConfiguration>
```

6. Run the *Data Load* utility command syntax with the *dataload.bat* tool which is located in `workspace\bin` of the RAD environment. Give the absolute path to the `wc-dataload.xml` file. The call might look as follows:

```
..\bin\dataload.bat [path_to_your_dataload]\wc-dataload.xml
```

Load the custom access control policies for the CoreMedia Widget

1. Stop the IBM WebSphere Commerce server in the IBM RAD environment.
2. Copy the custom access control policies files `workspace/DataLoad/acp/common/CoreMediaContentDisplay.xml` and `workspace/DataLoad/acp/common/CoreMediaMicroSite.xml` to the access control policies directory which is located in `xml\policies\xml` of the RAD environment.
3. Run the *ACP Load* utility with the *acpload.bat* tool which is located in `workspace\bin` of the RAD environment. Give the absolute path to the `acp-file name.xml` file. The call might look as follows:

```
..\bin\acpload.bat [path_to_your_acp_dir]\acp-filename.xml
```

The *acpload* documentation can be found here: https://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/refs/rax-acpload.htm.



The *acpload* tool itself does not report any problems. So, check if the tool created 2 new XML files with the suffixes `_xmltrans.xml` and `_idres.xml` in `..\xml\policies\xml` for each policy file. Also look into `..\logs\acpload.log` and `..\logs\messages.txt` for errors.

Add the Widget UI to the Management Center app

1. Merge the content of the file `PageLayoutExtensionsLibrary.lzx` from the `WEBDEV/workspace/LOBTools/WebContent/WEB-INF/src/lzx/commerce/pagelayout` folder into the corresponding file

`PageLayoutExtensionsLibrary.lzx` in the WCS workspace. It is only one line of XML code.

2. Remove the file `PageLayoutExtensionsLibrary.lzx` from the Workspace archive and copy the `LOBTools` folder content into the `LOBTools` folder of the IBM RAD workspace.

Copy the Stores Folder and Apply JSP Customizations

Copy and merge the content of the `Stores/` folder of the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive into the IBM RAD workspace folder `Stores/` as described in [Section 3.4.10, “Customizing IBM WebSphere Commerce JSPs” \[72\]](#)

Using Placeholder Resolution for Asset URLs

If you have licensed *CoreMedia Advanced Asset Management* you can use placeholders for the CMS host and the store ID in your image URLs. [Section “Placeholder Resolution for Asset URLs” \[452\]](#) describes further details and how you enable placeholder resolution.

Refresh and Rebuild the workspace in Eclipse (RAD)

Now you have to refresh and rebuild the IBM workspace in the IBM RAD environment.

1. Refresh the projects in the IBM RAD system so that the new files are recognized:
 - a. Select the `Stores` project and press **F5**
 - b. Select the `WebSphereCommerceServerExtensionsLogic` project and press **F5**
 - c. Select the `LOBTools` project and press **F5**
2. Rebuild the `LOBTools`:
 - a. Rebuild the `LOBTools` in order to apply the changes to the management Center application.
 - b. Right-click the `LOBTools` project and select **Build OpenLazlo Project** from the context menu.

This steps might take some time.

3. Republish the WCS Server workspace in order to apply the changes to the shop web application. In the server view (bottom left corner) right click on the server instance and select **Publish** from the context menu.

You have updated the Management Center tools and the development workspace and the WCS server has been restarted.

3.4.12 Setting up SEO URLs for CoreMedia Pages

IBM WCS contains a default SEO-URL configuration for its shopping pages, such as product detail pages or category landing page. For a seamless integration of CoreMedia content pages like CoreMedia article pages or microsites the SEO-URL configuration needs to be extended. The *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive comes with a SEO-URL configuration, which you can apply to your project WCS workspace.

The CoreMedia SEO-URL configuration is required for the usage of CoreMedia Microsites and CoreMedia Content Display in your WCS environment.

As a prerequisite, SEO URLs require the custom access control policies, installed in [Section 3.4.11, “Deploying the CoreMedia Widgets” \[73\]](#).

In order to enable the CoreMedia SEO URLs do the following steps:

1. Define the SEO pattern and its mapping for a given StoreName (Aurora or AuroraEsite, for instance). See the IBM documentation at https://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.seositemap.doc/concepts/csdSEOpatternfiles.htm for more details about SEO configuration.

To do so, copy the SEO pattern file `workspace/Stores/WebContent/WEB-INF/xml/seo/stores/{StoreName}/SEOURLPatterns-CoreMedia.xml` to your project workspace.

For development, create a file `.reload` (text file) in the same directory and add this line: `reloadinterval = 30`. This will reload the SEO patterns file every 30 seconds.



2. Configure the handling of SEO Requests as follows:

Apply the Struts configuration from `workspace/Stores/WebContent/WEB-INF/struts-config-lc2.xml` from the CoreMedia archive to your project workspace. Do not forget to change storeIDs to your needs. The storeID is the number at the end of the values of the `name` attributes.

Check, that the Struts configuration is already referenced from the `init-param` with name "config" in your IBM WCS `web.xml` file. Otherwise, copy the configuration from the `workspace/Stores/WebContent/WEB-INF/web.xml` file.

3. Check if the copied JSP files already contain the parameter `externalSeoSegment`:

The SEO pattern specifies that the path segment after `/cm/` or `/microsite/` will be mapped to a JSP parameter `externalSeoSegment`. Make sure the parameter is actually recognized and prepared to be passed to the `lc-include` tag as `lc_externalRef` parameter.

```
<c:if test="${not empty param.externalSeoSegment}">
  <c:set var="lc_externalRef"
value="cm-seosegment:${param.externalSeoSegment}"/>
</c:if>
```

Otherwise, check the JSP files in the CoreMedia archive file and copy the settings to the JSPs in the IBM workspace.

4. Check SEO links

As defined in `SEOURLPatterns-CoreMedia.xml` the URL pattern *CoreMediaContentURL* and *CoreMediaMicroSiteURL* can be used from within the IBM `wcf:url` tag. You can find the implementation of URL generation for CoreMedia content with this tag in the JSP file `WCDE-ZIP/workspace/Stores/WebContent/Widgets-CoreMedia/com.coremedia.commerce.store.widgets.CoreMediaContentWidget/impl/templates/Content.url.jsp`. Check that this file is already included in your IBM workspace. Otherwise, copy it.

3.4.13 Event-based Commerce Cache Invalidation

CoreMedia DXP 8 integrates with *IBM WCS* without importing catalog data into the CMS. Since all catalog data is requested dynamically from the *IBM WCS* system, *CoreMedia DXP 8* comes with its own caching layer to provide fast access to *IBM WCS* data. In order to promptly reflect any changes of *IBM WCS* data, *CoreMedia DXP 8* supports event based cache invalidation. Apply the following changes to enable event based cache invalidation:

1. Add the `CacheInvalidationHandler` as described in [Section 3.4.7, “Configuring REST Handlers” \[67\]](#)
2. Add database triggers for cache invalidation of segments and marketing spots. In order to create corresponding entries in *IBM WCS CACHEIVL* table (where Dynacache Invalidation events are stored), several database triggers are needed. These are database specific but *IBM WCS* already provides correct working examples in the *WCS* workspace. You do not need all the sample triggers but only those for Segments and Marketing Spots.

Copy the triggers for the INSERT, UPDATE and DELETE cases for the following tables:

➔ EMSPOT

➔ MBRGRP

and add them to your database (for example, by using `http://HOST/webapp/wcs/admin/servlet/db.jsp`)

The sample files are located in the following directories:

- ➔ Oracle:
WCDE-ZIP\schema\oracle\cm.wcs.cacheivl.trigger.sql
- ➔ IBM DB2:
WCDE-ZIP\schema\db2\cm.wcs.cacheivl.trigger.sql
- ➔ Cloudscape/Derby:
WCDE-ZIP\schema\clouscape\cm.wcs.cacheivl.trigger.sql

If enabled IBM workspaces in your environment setup, you need to create these triggers for each workspace. If you enabled 5 workspaces for example, you need to create these triggers 5 times. Examples are given in the mentioned example files.



3.4.14 Deploying the CoreMedia Catalog Data

The Sample archive file contains CoreMedia store data that can be used together with the CoreMedia CMS Blueprint demo data. Part of the data can be imported via SAR files, the other via data load.

Publishing SAR Files

This content can be found in the following folders below the `sar/` folder:

- ➔ `esite-base`
- ➔ `esite-base-ws`
- ➔ `esite-marketing`
- ➔ `CMSites`

You have to publish these data into the WCS as described below. See the IBM documentation at https://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.admin.doc/tasks/tpbpbst.htm for more details.

1. Start the WCS server.
2. Pack the `SAR-INF` and `WEB-INF` subfolders of `CMSites` into a Zip file `cmsites.sar`. Change the file extension of the Zip file to ".sar".
3. Publish the `cmsite.sar` file to the WCS, following the IBM documentation.
4. Adapt the database settings in the environment configuration files `wc-data-load-env.xml` and `wc-dataload-env-ws.xml` of each `esite` (not for `cmsite`) to the values of your database. You will find the files below: `sar/<siteName>/WEB-INF/stores/PerfectChef_Catalog[_WS]/data/dataload.`

5. In the files `sar/esite-base/WEB-INF/stores/PerfectChef_Catalog/data/ibm-wc-load.xml` and `sar/esite-base/ws/WEB-INF/stores/PerfectChef_Catalog_WS/data/ibm-wc-load.xml` adapt the links to the location of the other SAR files in the task element with name "sarFileDeploy".

```
<task name="sarFileDeploy">
  <param name="storeArchiveFilename"
value="/home/wcuser/sars/esite-marketing.sar"/>
```

Example 3.5. Default link setting

6. Pack the SAR-INF and WEB-INF subfolders of each esite folder into a Zip file named after the esite folder. For example, `esite-base.zip`. Change the file extension of the three Zip files to ".sar".
7. Publish the `esite-base.sar` into the WCS, using the *publishstore* tool. The other two files are automatically imported.

Importing Data via Data Load

See the IBM WebSphere Commerce documentation https://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.data.doc/concepts/cm-lbatchoverview.htm for more details about data load.

1. Stop the IBM WebSphere Commerce server in the IBM RAD environment.
2. Adapt the database settings in the *Data Load* environment configuration files (`SAMPLEDATA-ZIP\workspace\DataLoad\dataload\common\wc-data-load-env[-<siteName>].xml`) from the *Sample archive* Zip file to the settings of your WebSphere database.

You can retrieve your database settings from the IBM RAD environment configuration file `WC/xml/config/wc-server.xml`, at the following XML element:

```
<InstanceProperties>
  <Database>
    <DB>
```

3. Use the *Data Load* utility to load the data for all sites. Give the absolute path to the `wc-dataload.xml` file, for example `c:\lc-demo-data\workspace\DataLoad\dataload\common\AuroraESite\wc-dataload.xml`.

3.4.15 Troubleshooting

Problem

You get an error `com.ibm.commerce.catalog.facade.client.CatalogNavigationViewException` on the Commerce server or the following error on the Solr server:


```
SolrCore E org.apache.solr.common.SolrException log
org.apache.solr.common.SolrException:
  org.apache.lucene.queryParser.ParseException:
    Cannot parse 'catentry_id:"123456" ....)': too many boolean
    clauses
```

Possible cause

You have a large number of SKUs per product during product entitlement.

Possible solution

Increase the `MaxBooleanClause` property in `solrconfig.xml` to 8192. Keep in mind, that each index has its own `solrconfig.xml` file with the `maxBooleanClauses` setting.

3.5 Using the CoreMedia Workspace

This section describes how you can use the workspace:

- [Section 3.5.1, “Building the Workspace” \[82\]](#) describes how you can build the workspace for the first time.
- [Section 3.5.2, “Working With the Box” \[83\]](#) describes how you start and manage the virtualized CoreMedia environment based on VirtualBox, which is the recommended way.
- [Section 3.5.3, “Locally Starting the Components” \[85\]](#) describes how you start the CoreMedia components directly on your developer machine.
- [Section 3.5.5, “Developing with Components and Boxes” \[98\]](#) describes how you combine a virtualized environment with local components for development.
- [Section 3.5.6, “Developing Against a Remote Environment” \[101\]](#) describes how all of your developers can develop against a common remote environment.

3.5.1 Building the Workspace

The *CoreMedia Blueprint* workspace contains a complete CoreMedia CMS system. The first step is to build the system and to start all components or boxes. This enables you to test your changes and new features locally, which is a convenient setup for a new project.

In this manual `$CM8_BLUEPRINT_HOME` will refer to the root directory of the *Blueprint* workspace. If you set it as an environment variable, you can simply copy and paste the command line snippets. Otherwise, you have to substitute paths accordingly.

Go to the *Blueprint* workspace and build the system with the following commands:

```
$ cd $CM8_BLUEPRINT_HOME
$ mvn clean install
```

If you want to build without tests, add the `-DskipTests` option to the Maven call.

If you want to prepare your build environment to be independent of any network resources, you can achieve this by calling `mvn dependency:go-offline`. After this call has succeeded you can build offline by adding the option `-o` or `--offline` to your Maven calls.





Maven offers many command line options to improve efficiency by building only the required subset of modules. Especially the `-pl`, the `-am`, the `-amd` and the `-rf` options are interesting for your daily work. If, for example, you want to build all modules affected by your change in module X, you simply need to call `mvn -pl ":X" -amd clean install` from the root directory of the *Blueprint* workspace. If you don't want the dependent packages to be build, simply execute that call from within the `modules` directory.

If your build was successful, you can proceed with the next section. For a reference about all *Maven* build profiles, see [Section 9.5, “Maven Profile Reference” \[475\]](#)

3.5.2 Working With the Box

In this section you will learn how to start virtual machines based on CentOS in Virtual Box and install the *CoreMedia Blueprint* services and applications onto these machines using the RPM artifacts being built by the modules below packages.

Starting the Box

During the build of the workspace, you have prepared an RPM repository below `$CM8_BLUEPRINT_HOME/boxes/target/shared/rpm-repo`. All files below the `shared` folder are accessible from the box.

Start the box with the following command:

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant up
```

If you do this the first time, you will notice a lengthy download of a large file. This is the download of the base box, a bare CentOS 6 system with only Java being installed. This download will only be done once as Vagrant is caching the base boxes.

Download of the base box takes time

After the box has been downloaded Vagrant imports the box, configures it, for example sets the number of CPUs or the RAM and the network configuration. The latter step requires you to add a network adapter to the host system, so you have to confirm some dialogues if you are using Windows.

After configuring the hardware, Vagrant boots up the box. When the system is running the *vagrant-vbguest* and the *vagrant-omnibus* Vagrant plugins will now outfit the box with the matching *Virtual Box Guest Additions* and the configured version of *Chef*. These steps may take a couple of minutes but guarantee you a proper host to guest communication and eases the maintenance of the base boxes.

Now the box is ready for provisioning and *Vagrant* will start *Chef* to install and configure CoreMedia services. Some minutes later your blueprint box will be online

with all services running, all content imported and published and ready to develop against.

For an overview of the installed applications and services and their links, see the `README.md` markdown file, in the root directory of the workspace. To see an overview of all ports see [Section 9.1, “Port Reference” \[463\]](#).

Installed applications

If you want some insight on the state of your system, you can open up the *PSDash* web interface at <http://localhost:8999> to monitor available log files. This interface is only available on the box started with Vagrant and is intended for development support only. To log into the box simply execute

Check system status

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant ssh
```

If you don't have `ssh` on your `PATH`, you can call

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant ssh-config
```

and use the configuration with the `ssh` tool of your choice.

Depending on the performance of the host machine, the process of starting up and provisioning the management box can take some time. To avoid this overhead, you can pause the system until you need it again. From the moment the system is up and running, it's not Vagrant running the system anymore, it's Virtual Box and you may use all features of Virtual Box to control your box, for instance export it, pause it or resume it. Vagrant is nothing more than a configuration and bootstrapping tool.

To connect to the WebDAV web application as a network drive a necessary certificate will be generated automatically during the provisioning. How to install such a certificate is described in [Section 8.6, “WebDAV Support” \[442\]](#).

WebDAV

By default, the delivery *Blueprint* CAE will be connected with the *Master Live Server* not the *Replication Live Server*.

Update your Box

If you have developed a feature in module *X* and you want to deploy and test it in your isolated box environment, all you have to do is the following:

1. Build the changed module and all affected artifacts, for the `contentserver-blueprint-component`, for example, execute:

```
$ cd $CM_BLUEPRINT_HOME/modules
$ mvn -pl ":contentserver-blueprint-component" clean install -amd
```

2. Update the RPMs in the boxes module:

```
$ cd $CM_BLUEPRINT_HOME/boxes
$ mvn antrun:run
```

3. Reprovision the box containing the change, for example `vagrant provision`, to update the RPMs. You can also use the provision call without new RPMs, just to apply any configuration changes. Only the services for which Chef detects changes, will be restarted.

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant provision
```

Suspend your Box

When you want to suspend your work for some reasons for example your daily job is done, you can simply suspend the box calling the `suspend` command on a box for instance `vagrant suspend`.

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant suspend
```

Resume your Box

To resume your work, simply call the `resume` command on a box for example `vagrant resume`.

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant resume
```

Destroy your Box

When you have finished your work or you want to start on from scratch, simply call the `destroy` command on a box, for example `vagrant destroy`.

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant destroy
```

3.5.3 Locally Starting the Components

[Section 3.5.2, “Working With the Box” \[83\]](#) describes how you start the virtualized environment. This section will show you how to start the underlying components directly on your developer machine and how to import the demo content. To achieve this you need to configure the databases as described in [Section 3.3.5, “Configuring Local Setup” \[53\]](#). This section will provide you with the command line snippets to start the components, but you can also run Maven commands from your IDE, if you don't like switching between applications.



For the very initial development purposes or for debugging, the server components can be started using the way described in this section, but the recommended way to start server components is to use the virtualized setup provided with the management box. It provides you with a far more reproducible state and prevents you from polluting your developer machine with project specific infrastructure and state.

Furthermore, some basic knowledge about the *CoreMedia CMS* architecture and components, especially the Content Server, the CAE and CoreMedia Studio would be helpful to cope with this section.

If you want to start developing right away, you should continue with [Section 3.5.5, “Developing with Components and Boxes” \[98\]](#).

Starting Management Components

Starting the Search Engine

The *Content Management Server* needs the *Search Engine* to index all documents for full text search within *CoreMedia Studio*.

```
$ cd $CM_BLUEPRINT_HOME/modules/search/solr-webapp
$ mvn tomcat7:run-war
```

When you read the message that Tomcat server is up and running, simply test the *Search Engine* by browsing to its administration page at <http://localhost:44080/solr>. If you can see the administration page, everything is fine.

Starting the Content Servers

Now you start the *Content Management Server*, the backbone of a *CoreMedia CMS*.

```
$ cd
$CM_BLUEPRINT_HOME/modules/server/content-management-server-webapp
$ mvn tomcat7:run-war
```

When you read the message that Tomcat server is up and running, check the IOR URL at <http://localhost:41080/coremedia/ior>. When you get an IOR like the following,

```
IOR:000000000000003849444c3a686f782f636f72656d2f636f7262612f4c6f676
96e536572766963655769746850726f746f636f6c56657273696f6e3a312e3000
000000010000000000000082000102000000000a3132372e302e302e3100be2b00
000031afabcb00000000207ba1b4de0000000100000000000000100000008526f
6f74504f41000000008000000030000000014000000000000200000001000000
200000000000010001000000020501000100010020000101090000000100010100
000000260000000020002
```

the *Content Management Server* is running.



If the Content Management Server fails to start up, you probably did not provide a valid license or you haven't configured it yet. See [section “Configuring the licenses” \[47\]](#) and apply the necessary configuration.

You can start the *Master Live Server* the same way and check its IOR at port 42080. (However, this is no prerequisite for the *Preview CAE*, and you can also do it later.)

```
$ cd $CM_BLUEPRINT_HOME/modules/server/master-live-server-webapp
$ mvn tomcat7:run-war
```

Importing the Blueprint contents

In order to be able to explore the Blueprint features, CoreMedia provides sample content, which you can import into the *Content Management Server*. During the build of the workspace the sample content and the users of all activated extensions are collected into one Zip file. Take the file `boxes/target/shared/content/content-users.zip` and unzip it into a folder. Now, import the content with the following command, replace `$PATH_TO_CONTENT` with the path to the unzipped content `<PathToUnzippedFile>/content`:

```
$ cd
$CM_BLUEPRINT_HOME/modules/cmd-tools/cms-tools-application/target/cms-tools
$ ./bin/cm serverimport -u admin -p admin -r $PATH_TO_CONTENT

or, when you are using a 64-bit Windows

$ ./bin/cm64 serverimport -u admin -p admin -r $PATH_TO_CONTENT
```

This will import all the XML files you can find under the `content` folder.

Importing the Blueprint users

CoreMedia provides users for the demo content, which you can import to the *Content Management Server*. The users are also part of the Zip file `content-users.zip` for the demo content. You might find several XML files for several extensions. Replace `$PATH_TO_CONTENT` with the path to the demo user file, that is `<PathToUnzippedFile>/users/<filename>.xml`:

```
$ cd
$CM_BLUEPRINT_HOME/modules/cmd-tools/cms-tools-application/target/cms-tools
$ ./bin/cm restoreusers -u admin -p admin -f $PATH_TO_USERS

or, when you are using a 64-bit Windows

$ ./bin/cm64 restoreusers -u admin -p admin -f $PATH_TO_USERS
```

This will import all the users defined in the XML file. Import the other user files in the `users` folder similarly.

Starting the Content Feeder

Now that the *Content Management Server* and the *Search Engine* are up and running, you can start the *Content Feeder*.

```
$ cd $CM_BLUEPRINT_HOME/modules/search/content-feeder-webapp
$ mvn tomcat7:run
```

Afterwards you can check the *Content Feeder's* administration page at <http://localhost:39080/feeder/admin>. The *Content Feeder* feeds content (documents) to the *Search Engine* and will immediately start feeding. To check it, browse to the Solr administration page at <http://localhost:44080/solr> and search for `id:coremedia*`. This query searches for all index entries whose id starts with `coremedia`. If Solr finds some, the *Content Feeder* works properly.

Starting the Preview CAE Feeder

Like the *Content Feeder* the *CAE Feeder* depends on the *Content Management Server* and the *Search Engine* to be running. The *CAE Feeder* feeds content beans to the *Search Engine*.

```
$ cd $CM_BLUEPRINT_HOME/modules/search/caefeeder-preview-webapp
$ mvn tomcat7:run
```

Afterwards take a look into the *CAE Feeder's* logfile at `caefeeder-preview-webapp/target/logs/caefeeder.log`. The *CAE Feeder* should start feeding all the content beans based upon the content you have just imported into the *Content Management Server*. To check it, browse to the Solr administration page at <http://localhost:44080/solr> and search for `id:contentbean*`. This query searches for all index entries whose id starts with `contentbean`. If Solr finds some, the *CAE Feeder* works properly.

Starting the Workflow Server

The next component to start is the *Workflow Server*.

```
$ cd $CM_BLUEPRINT_HOME/modules/server/workflow-server-webapp
$ mvn tomcat7:run-war
```

When Tomcat is running, look into `workflow-server-webapp/target/logs/workflow.log`. If the logfile shows no errors and ends with

```
Server - Server: started (Workflow Server Starter)
```

the *Workflow Server* is running.

Starting the Editorial Components

Starting the Preview CAE

Next, you can start the preview CAE. That is the CAE which will be used by the editors for checking their written content.

```
$ cd $CM_BLUEPRINT_HOME/modules/cae/cae-preview-webapp
$ mvn tomcat7:run -Pdevelopment-ports
```

When the web application is up and running browse to <http://preview-corporate.localhost:40081/> to see the Blueprint demo web application. If the URL cannot be resolved, you have probably not added the fake domains to your `/etc/hosts` file, see [Section 3.3.4, “Configuring Vagrant Based Setup” \[50\]](#) for the list of required host mappings. If the page looks broken, your browser is possibly configured too restrictive with an anti-scripting plugin. Trust CoreMedia, and grant the Blueprint web application all required permissions. You should be able to click through the whole web application, register yourself, login to rate and comment on various content.

In a production deployment the Preview CAE and Studio run in the same Tomcat instance and thus have the same AJP and HTTP ports. If you use the tomcat7 Maven plugin however, each web application runs in a separate Tomcat instance, so you would encounter a port clash between the Preview CAE and Studio. The `development-ports` profile overrides the ports for the preview CAE and prevents a port clash.

Starting CoreMedia User Changes Web Application

After the *Content Management Server* is running, you can start the *User Changes* web application.

Start *CoreMedia User Changes* web application with the Tomcat plugin:

```
$ cd $CM_BLUEPRINT_HOME/modules/server/user-changes-webapp
$ mvn tomcat7:run
```

When Tomcat is running, look into `user-changes-webapp/target/logs/user-changes.log`. If the logfile shows no errors and it contains

```
Initializing user-change-listener
Attach user-change-listener to content repository with timestamp
```

the *User Changes* web application is running.

Starting CoreMedia Studio

Now, the *Content Management Server*, the *Workflow Server* and the *Preview CAE* are running.

If you skipped the *Master Live Server* before, you should start it now, because the next application to start is *CoreMedia Studio*, and without the *Master Live Server* you would not be able to publish content.

Start *CoreMedia Studio* with the Tomcat plugin:

```
$ cd $CM_BLUEPRINT_HOME/modules/studio/studio-webapp
$ mvn tomcat7:run -Pdevelopment-ports
```

After the Tomcat server started up, you can browse to <http://localhost:40080/> to open *CoreMedia Studio*. Login as user "admin" with password "admin".

Starting the WebDAV Server

The *WebDAV Server* is an optional web application for editing content via the WebDAV protocol. You can start it using the Tomcat plugin as follows:

```
$ cd $CM_BLUEPRINT_HOME/modules/editor-components/webdav-webapp
$ mvn tomcat7:run-war
```

WebDAV clients can now connect to <https://localhost:8086/webdav>. Open the URL with your browser, login as admin/admin, and you can browse through the repository and find some documents, especially pictures. With Microsoft Windows 7 enter the following command to create a network drive that is connected to the *WebDAV Server*:

```
$ net use * https://localhost:8086/webdav/ * /user:admin
/persistent:no
```

Starting the Site Manager

For user management and special content editing tasks which are not yet covered by *CoreMedia Studio* you can use the *Site Manager* (formerly known as *CoreMedia Editor*).

Note, that the *Site Manager* is only supported with a 32bit Java. You can set the used Java in the `bin/pre-config.jpif` file with the `JAVA_HOME` property. By default, it is set to your local `JAVA_HOME` environment variable.



```
$ cd
$CM_BLUEPRINT_HOME/modules/editor-components/editor/target/editor
$ bin/cm editor
```

Starting the other Components

Some components from the *Delivery Environment* are left out, but they are not handled differently than the components started before. [Table 3.5, “Modules in the Workspace” \[91\]](#) lists those components. If you want to use the *Delivery Environment*, you have to publish the demo content from the *Management Environment*

to the *Delivery Environment*. Otherwise, you will not see any content in the live CAE. You can use the `bulkpublish` tool.

```
$ cd
$CM BLUEPRINT_HOME/modules/cmd-tools/cms-tools-application/target/cms-tools
$ ./bin/cm bulkpublish -u admin -p admin -a -b -c

or, when you are using a 64-bit Windows

$ ./bin/cm64 bulkpublish -u admin -p admin -a -b -c
```

Component	Location	Description
Master Live Server	modules/server/master-live-server-webapp	The <i>Master Live Server</i> is started exactly the same way as the <i>Content Management Server</i>
Replication Live Server	modules/server/replication-live-server-webapp	See the <i>Master Live Server</i> and the <i>Content Management Server</i>
Live CAE Feeder	modules/feeder/caefeed-er-live-webapp	The <i>CAE Feeder</i> for the live CAEs, started the same way as the preview <i>CAE Feeder</i> .
Live CAE	modules/cae/cae-live-webapp	The live CAE is started the same way as any other web application within the development workspace.

Table 3.5. Modules in the Workspace

3.5.4 Developing with Apache (optional for e-Commerce)

For some PerfectChef features CoreMedia makes use of the Apache HTTP server (see [Section 5.4, “Connecting with an IBM WCS Shop” \[207\]](#) for details), especially rewrite rules and handling of HTTPS requests (which is why Login does not work with a standalone Tomcat). The triangular communication between CAE, WCS and the client raises some more low level technical constraints concerning the cookie domain and cross domain Ajax requests. Without going into detail here, the easiest way to solve all such problems flexibly during development is to hide the CAE and the WCS behind a common Apache proxy. Moreover, according to the production setup you should test your development with Apache anyway to make sure that your features work correctly. Since Apache configuration is a tedious task, CoreMedia provides a virtual machine with a preconfigured Apache server, which you can easily use and maintain.

Since CAE and WCS invoke each other mutually, there must be a 1:1 relationship between them, otherwise some LiveContext features will not work correctly. This means you would need two WCS instances for your Preview and Live CAE. However, a WCS instance needs quite some resources, and for most development use cases it is feasible to dispense a fully featured system and work with a shared WCS in-

stance for the Preview and the Live CAE. Therefore, this setup refers to a single WCS instance.

Prerequisites

You need to have installed Vagrant and VirtualBox as described in [section “Developer Setup” \[40\]](#). You also need a local instance of IBM WebSphere Commerce as described in [Section 3.4, “Customizing IBM WebSphere Commerce” \[58\]](#).

The Apache server on the Vagrant virtual machine connects via the following ports:

- 49009 - via AJP to the Live CAE on your local host
- 40010 - to the Preview CAE
- 40009 - to *Studio*

Open these ports in your firewall, that is for requests from 192.168.252.100 to local host.

Switch off blocker plugins in your browser for *CoreMedia Blueprint* applications. Otherwise, some JavaScript code might not be executed, and the pages will be incomplete.

Building the Web Applications

1. You need two Maven profiles to build the *Blueprint* web applications according to the Apache setup. This concerns mainly the domains for absolute URLs. Add the profiles to your `.m2/settings.xml` file:

```
<profile>
  <id>localEnvironment</id>
  <properties>
    <cae.is.standalone>false</cae.is.standalone>
    <livecontext.ibm.wcs.host>shop-ref.ecommerce.mycompany.com</livecontext.ibm.wcs.host>
    <livecontext.apache.wcs.redirect.host>shop-helios.blueprint-box.vagrant</livecontext.apache.wcs.redirect.host>
    <blueprint.site.mapping.helios>helios.blueprint-box.vagrant</blueprint.site.mapping.helios>
    <livecontext.cookie.domain>blueprint-box.vagrant</livecontext.cookie.domain>
    <livecontext.ibm.wcs.store.id>perfectchef10851</livecontext.ibm.wcs.store.id>
    <livecontext.cookie.domain>blueprint-box.vagrant</livecontext.cookie.domain>
  </properties>
</profile>
<profile>
  <id>localPreviewEnvironment</id>
  <properties>
    <cae.is.standalone>true</cae.is.standalone>
    <livecontext.ibm.wcs.host>shop-ref.ecommerce.mycompany.com</livecontext.ibm.wcs.host>
    <livecontext.apache.wcs.redirect.host>shop-preview-helios.blueprint-box.vagrant</livecontext.apache.wcs.redirect.host>
    <studio.preview.url.prefix>blueprint/servlet</studio.preview.url.prefix>
    <blueprint.site.mapping.helios>preview-helios.blueprint-box.vagrant</blueprint.site.mapping.helios>
    <STUDIO_WEBAAPP_NAME>STUDIO_WEBAAPP_NAME</STUDIO_WEBAAPP_NAME>
    <STUDIO_ENVIRONMENT>development</STUDIO_ENVIRONMENT>
    <livecontext.ibm.wcs.store.id>perfectchef10851</livecontext.ibm.wcs.store.id>
    <livecontext.cookie.domain>blueprint-box.vagrant</livecontext.cookie.domain>
    <livecontext.ibm.wcs.store.id>perfectchef10851</livecontext.ibm.wcs.store.id>
    <livecontext.cookie.domain>blueprint-box.vagrant</livecontext.cookie.domain>
  </properties>
</profile>
<!-- copied from profile development-ports in the blueprint pom -->
<PREVIEW_AJP_PORT_SUFFIX>010</PREVIEW_AJP_PORT_SUFFIX>
<PREVIEW_HTTP_PORT_SUFFIX>081</PREVIEW_HTTP_PORT_SUFFIX>
</properties>
</profile>
```

2. Build the web applications (*Studio*, *Preview CAE* and *Live CAE*) and the RPMs for Apache as follows:

```
cd blueprint
mvn clean install -DskipTests -am -pl :cae-preview-webapp,:studio-webapp -PlocalPreviewEnvironment
mvn clean install -DskipTests -am -pl :cae-live-webapp -PlocalEnvironment
mvn clean install -am -pl :studio-apache -PlocalPreviewEnvironment
```

```
mvn clean install -am -pl :delivery-apache -PlocalEnvironment
cd boxes
mvn clean install -DskipTests
cd ..
```

3. Create a `.vagrantuser` file in `$CM_BLUEPRINT_HOME` with the following content:

```
yum:
  remi_repository_mirrorlist: "http://centos-mirror/rpms.famillecollet.com/enterprise/6/remi/mirror"
vm:
  memory: "1024"
  cpu: "1"
  tld: "blueprint-box.vagrant"
chef:
  run_list: "role[apache-only]"
```

4. Install and start the vagrant box with the following call. This will take a few minutes if you do it for the first time.

```
cd blueprint
vagrant up
```

Configuring the Network

Connect the following machines as described below:

- The Vagrant box running the Apache server
- The VMware clone of an IBM WCS RAD
- Your local *Studio* and *CAE*

VMware Clone of an IBM WCS RAD

1. Set the network connection of your VMware image to NAT mode. By default, it is configured to Bridged mode.
2. Open `C:\Windows\System32\drivers\etc\hosts` and define a mapping for the fragment suppliers as follows:

```
192.168.252.100    fragment.supplier.blueprint-box.vagrant
preview-fragment.supplier.blueprint-box.vagrant
```

The fragment suppliers are the hosts from which the Commerce Server tries to fetch CoreMedia fragments. By default, these are server aliases for virtual hosts of the Live and Preview CAE.

3. Open `C:\IBM\WCDE_ENT70\workspace\Stores\WebContent\WEB-INF\web.xml` and
 - a. change the cookie domain of `com.coremedia.fragmentConnector.cookieDomain` to `.blueprint-box.vagrant`

- b. change the value of `com.coremedia.fragmentConnector.liveCaeHost` to `http://fragment.supplier.blueprint-box.vagrant` and the value of `com.coremedia.fragmentConnector.previewCaeHost` to `http://preview-fragment.supplier.blueprint-box.vagrant`

4. Start the Commerce Server.

Vagrant Box containing the Apache Server

You have already started the Vagrant VM. Now you must add an entry to its `/etc/hosts` file and restart the Apache server. Just apply the following steps, replacing `<your-commerce-vm-ip>` by the IP address of your IBM Commerce VM. The exact `shop-ref` domain must be the same as the value of the `livecon.text.ibm.wcs.host` property in the `localEnvironment` Maven profile which you have already configured.

```
cd blueprint
vagrant ssh
sudo su -
echo "<your-commerce-vm-ip> shop-ref.ecommerce.mycompany.com" >> /etc/hosts
service httpd restart
```

Your Development Computer

You must add the Apache Vagrant VM and the IBM Commerce VM to your local `/etc/hosts` file, so that your browser can resolve the links of the *Blueprint* application.

```
sudo su -
echo "192.168.252.100 blueprint-box.vagrant helios.blueprint-box.vagrant
studio-helios.blueprint-box.vagrant \
preview-helios.blueprint-box.vagrant shop-helios.blueprint-box.vagrant
shop-preview-helios.blueprint-box.vagrant \
shop-preview-production-helios.blueprint-box.vagrant" >> /etc/hosts
echo "<your-commerce-vm-ip> shop-ref.ecommerce.mycompany.com" >> /etc/hosts
```

On a Windows machine the hosts file is `C:\Windows\System32\drivers\etc\hosts`

Now you can start the three Tomcat instances for Studio, Preview CAE and Delivery CAE:

```
cd blueprint/modules/cae/cae-preview-webapp
mvn tomcat7:run -PlocalPreviewEnvironment,<profile-to-your-content-server>

cd blueprint/modules/cae/cae-live-webapp
mvn tomcat7:run -PlocalEnvironment,<profile-to-your-content-server>

cd blueprint/modules/studio/studio-webapp
mvn tomcat7:run -PlocalPreviewEnvironment,<profile-to-your-content-server>
```

If all systems are up and running, you should visit each of the following URLs in order to install the SSL certificates into your browser. Otherwise, you will be bothered with the pop-ups later, and the Studio preview will not work properly at all.

- ➔ <https://helios.blueprint-box.vagrant/>
- ➔ <https://studio-helios.blueprint-box.vagrant/>

- ➔ <https://preview-helios.blueprint-box.vagrant/>
- ➔ <https://shop-preview-helios.blueprint-box.vagrant/webapp/wcs/stores/servlet/en/perfectchefesite>
- ➔ <https://shop-helios.blueprint-box.vagrant/webapp/wcs/stores/servlet/en/perfectchefesite>

Checklist

If you followed the instructions so far, all components and virtual machines are wired correctly. However, throughout the project lifecycle you possibly adapt the setup to your specific needs, invent some shortcuts for redeployment or modify the configuration of a component temporarily or permanently. Tracking down the indirections of machines referencing each other can be tedious, so here is a list of the relevant host configurations which you should check first in case of problems. The following table shows the involved machines with their default addresses as provided by the setup and the symbolic names used to reference the machines.

Component	IP, Port	Virtual Host Names	Scope
WCS	[The IP of your WCS host]	shop-ref.ecommerce.coremedia.com	CAE, Apache
Preview CAE	192.168.252.1:40010		Apache
Live CAE	192.168.252.1:49009		Apache
Apache	192.168.252.100	helios.blueprint-box.vagrant	public
		shop-helios.blueprint-box.vagrant	public
		preview-helios.blueprint-box.vagrant	editor
		shop-preview-helios.blueprint-box.vagrant	editor
		fragment.supplier.blueprint-box.vagrant	WCS

Table 3.6. Components of the Apache Development Setup

The concrete values may be adapted to your particular environment. Just make sure that you change it consistently at all of the following checkpoints. The Preview CAE and the Live CAE are supposed to run on your local machine. They do not need fancy symbolic names, because they are not exposed to user agents but only accessed internally by Apache. The network adapter with the mapping for 192.168.252.1 is set up by Vagrant. The IP address 192.168.252.100 of the Vagrant machine which hosts the Apache server is configured in the `blueprint/Vagrant` file. The symbolic names are Apache virtual hosts and server aliases. You find their configurations in the various `.conf` files under `/etc/httpd/conf.d`. Once the machines are up and running, you should check their connections as follows. The wiring of the machines includes some entries in their `/etc/hosts` files, be-

cause the names of development machines are usually not resolved by a DNS server.

WebSphere Commerce System

The WCS needs entries for `fragment.supplier.blueprint-box.vagrant` and `preview-fragment.supplier.blueprint-box.vagrant` in its `C:\Windows\System32\drivers\etc\hosts` file. The URL prefix of the CAE to fetch the fragments from is configured in `Stores\WebContent\WEB-INF\web.xml`:

```
<context-param>
  <description>
    This is the base for all fragment URLs for the preview site. It must point to a CoreMedia
    Preview-CAE. An empty value is allowed.
    In this case the liveCaeHost will be used instead and all fragments will be received from the
    Live-CAE.
  </description>
  <param-name>com.coremedia.fragmentConnector.previewCaeHost</param-name>
  <param-value>http://preview-fragment.supplier.blueprint-box.vagrant</param-value>
</context-param>
<context-param>
  <description>
    This is the base for all fragment URLs for the live site. It must point to a CoreMedia Live-CAE.
    This parameter is mandatory.
  </description>
  <param-name>com.coremedia.fragmentConnector.liveCaeHost</param-name>
  <param-value>http://fragment.supplier.blueprint-box.vagrant</param-value>
</context-param>
```

CAE

The CAE references the WCS via the `livecontext.ibm.wcs.host` property. Therefore, the CAE machine (that is, your local machine) needs a hosts entry for `shop-ref.ecommerce.coremedia.com`.

User Agent

You probably want to check the results of your development work in a browser, so your local machine does not only serve as CAE host but also as a user agent. In order to access the *CoreMedia DXP 8* applications, you must configure the Apache virtual host names `helios.blueprint-box.vagrant` and `shop-helios.blueprint-box.vagrant` for the Live application and the respective virtual hosts for the Preview application in your hosts file.

Apache

`shop-ref.ecommerce.coremedia.com` is referenced in various Apache configuration files, thus the Apache machine needs a hosts entry for it. The CAEs are referenced by their IP addresses.

The Apache configuration files, which are located under `/etc/httpd/conf.d`, declare virtual hosts for the names mentioned in [Table 3.6, “Components of the Apache Development Setup” \[95\]](#) and map them to the CAEs and the WCS. For example, the virtual host `shop-preview-helios.blueprint-box.vagrant` is configured in `commerce-shop-preview/commerce-shop-preview.conf` and looks like this:

```
<VirtualHost *:80>
  ServerName shop-preview-helios.blueprint-box.vagrant
  RequestHeader set X-FragmentHost preview
```



```

<Proxy balancer://commerce-shop-cluster>
  BalancerMember http://shop-ref.ecommerce.coremedia.com route=shopWorker loadfactor=1 ttl=300
</Proxy>
<Proxy balancer://cae-cluster>
  BalancerMember ajp://192.168.252.1:40010 route=studioWorker loadfactor=1 ttl=300
</Proxy>
...
</VirtualHost>
<VirtualHost *:443>
  ServerName shop-preview-helios.blueprint-box.vagrant
  RequestHeader set X-FragmentHost preview
  ...
</VirtualHost>

```

There are two virtual host declarations for each name, one for HTTP and one for HTTPS. The `RequestHeader` defines a context to identify from which CAE (Live or Preview) the fragments will be received.

Since this development setup uses a shared WCS for the Preview and Live CAEs, a request header is used to identify which CAE will be used to receive fragments. If the request header `X-FragmentHost` is set to `preview`, the fragments will be received from `preview-fragment.supplier.blueprint-box.vagrant`. If this request header is set to `live` or is not set, all fragments will be received from `fragment.supplier.blueprint-box.vagrant`. For catalog image delivery in a shared WCS environment see [Section “Placeholder Resolution for Asset URLs” \[452\]](#)

Developing New Apache Configurations

If you have successfully built your local development system as described above, you can change the Apache configuration.

```

cd blueprint
vagrant up          # or vagrant resume if you suspended it
mvn clean install -am -pl :studio-apache -PlocalPreviewEnvironment
mvn clean install -am -pl :delivery-apache -PlocalEnvironment
cd boxes
mvn antrun:run
cd ..
vagrant provision

```

For Linux *Blueprint* provides a script for this task: `workspace-configuration/apache/updateApaches.sh`. There are two more scripts which update only the Preview Apache or the Delivery Apache, respectively. You can also apply these scripts from IntelliJ Idea as follows:

1. Install the Bash Support Plugin
2. Create a new run configuration
3. Select the Bash Support Plugin
4. Reference the script you like and change name of the run configuration
5. Run it

3.5.5 Developing with Components and Boxes

By now, you should have learned how to build the workspace, start the box and optionally how to start the components directly from the workspace. In this section you will learn how to combine both for a nice reproducible development round-trip.

Overview of the Development Infrastructure

The required stack of services and components you need to start before you can start developing with *CoreMedia Studio* and the *Blueprint CAE* is considerably large. By using either a common remote development system or the virtualized box, you are being relieved from setting up and maintain those services.

Both options provide their advantages and disadvantages. The common remote environment is saving your developer machines hardware but since it is shared by all developers, it must be highly available and you have to be cautious on updates. The virtualized solution requires a large amount of RAM to be available on your developer machine but it keeps you isolated from other developers and gives you full control to deploy upgrades or control the services.

Development Round-trip using Virtualization

Virtualize intelligently

Before you start your virtualized environment, you should consider, what services you need in order to start your development applications from within your IDE. By default, the virtualized environment will not only contain those requirements, but also the applications you might want to change, such as *CoreMedia Studio* or *CAE*. This increases the startup time, the memory and the CPU footprint on your host machine. To decrease these factors, you can decrease the number of services started by choosing a different *Chef* role to run and then decrease the amount of memory reserved for the box.

To set a different *Chef* role and reduce the memory footprint, simply add a `.vagrantuser` file beside the `Vagrantfile` and apply the following YAML configuration to it:

```
vm:
  memory: "2048"
chef:
  run_list:
    - "role[base]"
    - "role[storage]"
    - "role[management]"
```

This will start the box with only 2048mb RAM and provision it only with the services required by *CoreMedia Studio* and the *CAE*. With

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant user parameters
```

you can print out all active configuration settings done with the *nugrant* plugin.

Build artifacts and fire up the box

Follow the steps below to set up your local isolated development system.

1. Build the complete workspace by executing a post-configured build. Since this should be the default, the required Maven call is:

```
$ cd $CM_BLUEPRINT_HOME
$ mvn clean install
```

2. Start the box by executing:

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant up
```

3. Start your *Studio* either from the command-line or from within your IDE. You can open Studio in your browser with <http://localhost:40080>. Make sure that in both cases, the *Maven* profile *vagrant* is being activated. Starting the web application from the command-line can be achieved by executing:

```
$ cd $CM_BLUEPRINT_HOME/modules/studio/studio-webapp
$ mvn tomcat7:run -Pvagrant
```

4. Start your preview CAE either from the command-line or from within your IDE. Make sure that in both cases, the *Maven* profile *vagrant* is being activated. Starting the web application from the command-line can be achieved by executing:

```
$ cd $CM_BLUEPRINT_HOME/modules/cae/cae-preview-webapp
$ mvn tomcat7:run -Pvagrant
```

Rebuild intelligently

If you have executed the above commands you are in the initial state to develop code for either *Studio* or *CAE*. You can minimize your round trip by using the appropriate Maven commands. Preferably you should make use of the reactor options of Maven to reduce the set of affected modules to be build.

- | | |
|---|---|
| -pl :<Maven artifactId> (project list) | Build a specific module from the workspace's root directory. |
| -am (also make) | Enforce Maven to build in advance all modules on which your current module depends. |

-rf :<Maven artifactId> (restart from)	Enforce Maven to start the build with the module given by <code>artifactId</code> .
-amd (also make dependencies)	Enforce Maven to also build all modules that depend directly or indirectly on your current module.

By combining the above parameters, you can minimize the set of required modules to build and therefore reduce the build time dramatically. The most common use case would be to use the parameters `-pl` in combination together with `-rf`.

If, for example, you have added some code to the `cae-base-lib` module and you want to restart the *preview-cae-webap* with the new code, all you need to do is, stop the *tomcat7-maven-plugin* and execute:

```
$ cd $CM_BLUEPRINT_HOME/modules
$ mvn -pl :cae-preView-webapp -rf :cae-base-lib clean install
```

If the build succeeded, you can simply restart the web application and your additional code will be active.

Reprovisioning Server Components intelligently

The initial provisioning of the box takes a lot of time and should therefore only be done rarely. Instead, you can reprovision your box only with those RPMs that contain your changes. To achieve this, the same Maven optimizations as shown above should be applied.

1. If for example you have added a new content type or altered an existing one and the schema migration can be done without resetting the database, you simply need to rebuild all affected modules and reprovision them to the box. To achieve this, you first need to rebuild the necessary *Maven* modules.

```
$ cd $CM_BLUEPRINT_HOME
$ mvn -pl :cms-tomCat,:mls-tomcat -rf :contentserver-blueprint-component clean install
```

In a second step you need to update the RPM repository folder at `$CM_BLUEPRINT_HOME/boxes/target/shared/rpm-repo` provided to your box by using the shared folder mechanism of *Virtual Box*. To achieve this you need to execute the following call:

```
$ cd $CM_BLUEPRINT_HOME/boxes
$ mvn antrun:run
```

This step copies all found RPM packages. So, building only what has changed is essential for minimal turnaround times.

2. To reprovision the changed RPMs to the box, you need to execute:

```
$ cd $CM_BLUEPRINT_HOME
$ vagrant provision
```

3. Now, you can restart your web applications or clients running on your developer machine to use the new features of your server components.

To prevent you from starting your boxes from scratch each time you start developing, you should get familiar with the additional *Vagrant* commands to suspend and resume boxes as described in [Section 3.5.2, “Working With the Box” \[83\]](#); Another interesting approach is to use the *sahara* plugin for *Vagrant*, it provides you with a sandbox functionality that allows you to either commit all happened changes or rollback to the last committed state. This is especially usefully, when you are developing features that alter the content repository and you want to rerun the feature on a constant repository state. In that case you can start a sandbox, provision your upgrades, run the test or tools, rollback and your round-trip is closed.



3.5.6 Developing Against a Remote Environment

Although the workspace is configured to use localhost everywhere, it is easy to develop against a remote environment. Either by changing localhost globally with the property `installation.host` in the project’s root POM file or by customizing some or all of the values listed in the table below. The property `installation.host` itself is just a convenience and should only be used within these specialized properties.

Property	Host of component
<code>cms.host</code>	Content Management Server
<code>mls.host</code>	Master Live Server
<code>rls.host</code>	Replication Live Server
<code>solr.host</code>	Solr search engine
<code>caefeeder-preview.host</code>	CAE Feeder (preview)
<code>caefeeder-live.host</code>	CAE Feeder (live)
<code>studio.host</code>	Studio
<code>mongo.db.host</code>	Mongo database
<code>database.host</code>	SQL database

Table 3.7. Environment properties

If you want to manage multiple environments, you can manage them in your `settings.xml` file:

Example 3.6. Adding Environments in `settings.xml`

```
<profile>
  <id>dev</id>
  <properties>
    <cms.host>cms.dev.host.com</cms.host>
    <mls.host>cms.dev.host.com</mls.host>
    <solr.host>cms.dev.host.com</solr.host>
    <database.host>db.dev.host.com</database.host>
    <mongo.db.host>mdb.dev.host.com</mongo.db.host>
  </properties>
</profile>
<profile>
  <id>retest</id>
  <properties>
    <cms.host>cms.retest.host.com</cms.host>
    <mls.host>cms.retest.host.com</mls.host>
    <solr.host>cms.retest.host.com</solr.host>
    <database.host>db.retest.host.com</database.host>
    <mongo.db.host>mdb.retest.host.com</mongo.db.host>
  </properties>
</profile>
<profile>
  <id>vagrant</id>
  <properties>
    <cms.host>blueprint</cms.host>
    <mls.host>blueprint</mls.host>
    <solr.host>blueprint</solr.host>
    <database.host>blueprint</database.host>
    <mongo.db.host>blueprint</mongo.db.host>
  </properties>
</profile>
```

You can choose now to connect to different environments by either manually adding a profile to your Maven call (such as `-Pdev`) or by setting the profile permanently in your `settings.xml` file.

Example 3.7. Activating Environment in `settings.xml`

```
<activeProfile>dev</activeProfile>
```

By default, a `vagrant` profile is provided for you, to develop against the virtualized *Vagrant* box.



4. Blueprint Workspace for Developers

CoreMedia Blueprint workspace is the result of CoreMedia's long year experience in customer projects. As *CoreMedia CMS* is a highly customizable product that you can adapt to your specific needs, the first thing you used to do when you started to work with *CoreMedia CMS* was to create a proper development environment on your own. *CoreMedia Blueprint* workspace addresses this challenge with a reference project in a predefined working environment that integrates all CoreMedia components and is ready for start.

CoreMedia Blueprint workspace provides you with an environment which is strictly based on today's de facto standard for managing and building Java projects by using Maven. That way, building your project artifacts is a matter of simply executing `mvn clean install`. Developers are able to test all the various *CoreMedia CMS* components directly within the same environment by executing `mvn tomcat7:run` and `mvn tomcat7:run-war` respectively. No further deployment is necessary.

Maven based environment

With the introduction of Chef as the provision tool of choice and Vagrant as the tool to prepare VirtualBox virtualized environments, setting up the server backend to start developing with *CoreMedia Blueprint* workspace is now a matter of minutes, rather than hours. A simple `vagrant up` management will provide you with all databases, content repository and search services without the need to install and configure any project specific software on the developer's system.

To achieve this simplicity all components within the *CoreMedia Blueprint* workspace are preconfigured with hostnames, ports, database schemes, users and passwords such that you don't need to worry about configuration while developing within the *CoreMedia Blueprint* workspace. Instead, right from the start, you may concentrate on the real work, on the business logic that creates your company's value in the first place.

Using a virtualized development infrastructure

CoreMedia Blueprint workspace creates RPM or Zip artifacts out of the box, which you can use for deployment. You can preconfigure or post-configure your components that means at build time or at installation time, respectively.

4.1 Concepts and Architecture

This chapter describes concepts and architecture of *CoreMedia Digital Experience Platform 8*.

- [Section 4.1.1, “Maven Concepts” \[104\]](#) describes how the Maven concepts are implemented within the *CoreMedia Blueprint* workspace.
- [Section 4.1.3, “Application Architecture” \[107\]](#) describes how CoreMedia applications are build from library and component artifacts and how deployable artifacts are build with package artifacts.
- [Section 4.1.4, “Structure of the Workspace” \[112\]](#) describes the folder structure of the *CoreMedia Blueprint* workspace.
- [Section 4.1.5, “Project Extensions” \[114\]](#) describes the extensions mechanism which lets you enable and disable extensions in one single location.
- [Section 4.1.6, “Virtualization and Provisioning” \[119\]](#) describes the virtualization and provisioning infrastructure based on Chef and Vagrant.

4.1.1 Maven Concepts

The *Maven* build and dependency system is the foundation of the *CoreMedia Blueprint* workspace. This section will introduce you into the concepts CoreMedia used with *Maven* to provide you with the best development experience as possible.

Packaging Types

By default, *Maven* provides you with several packaging types. The most important ones are the `pom`, `jar` and the `war` type. They should be sufficient for the most common kinds of development modules but whenever you try to either support proprietary formats or try to break whole new ground, those three packaging types aren't sufficient. Using only the `pom` packaging type together with custom executions of arbitrary plugins, gives you flexibility but adding and maintaining your `pom.xml` files is going to be a complex and costly process.

To reduce complexity, but even more important to enforce standards, CoreMedia came up with two custom tailored packaging types for the *CoreMedia Blueprint* workspace. The `coremedia-application` and the `jangaroo` packaging type. The `coremedia-application` type provides a build lifecycle and dependency profile for a proprietary application format, whereas the `jangaroo` type provides both for the *JavaScript* tool chain to build sophisticated and compelling user interfaces like *CoreMedia Studio*.

coremedia-application

The `coremedia-application` packaging type is provided by the `coremedia-application-maven-plugin`. When you take a look at the root `pom.xml` and search for this plugin, you will find two occurrences, one in the `pluginManagement` section and one in the `build` section. The latter definition contains the line `<extensions>true</extensions>` within its plugin body, telling *Maven* that it extends *Maven* functionality. In this case, *Maven* will register the custom lifecycle bound to the custom packaging type.

```
<plugin>
  <groupId>com.coremedia.maven</groupId>
  <artifactId>coremedia-application-maven-plugin</artifactId>
  <extensions>true</extensions>
</plugin>
```

Besides lifecycle, a custom packaging type can also influence if *Maven* dependencies of this type have transitive dependencies or not. Because CoreMedia wanted to keep the `coremedia-application` packaging type to be the pendant of the `war` packaging type, it does not have transitive dependencies either. For your modules to depend on other `coremedia-application` modules and their dependencies as well, this means, that you need to define an additional dependency to the same GAV (`groupId`, `artifactId`, `version`) coordinates but with packaging type `pom`.

```
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>probedog-application</artifactId>
  <type>coremedia-application</type>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>probedog-application</artifactId>
  <type>pom</type>
  <scope>runtime</scope>
</dependency>
```

Example 4.1. Dependencies for a CoreMedia application

You may know this pattern from working with `war` overlays if they are skinny too, which means that they contain no further versioned artifacts.

For further information about the `coremedia-application-maven-plugin`, you should visit the plugins documentation site at [CoreMedia Application Plugin](#).

Jangaroo

Like the `coremedia-application-maven-plugin`, the `jangaroo-maven-plugin` defines a *Maven* extension with a custom packaging type `jangaroo`, which introduces a custom lifecycle. This lifecycle binds the standard phases known from the `jar` packaging type to the corresponding goals for compiling ActionScript to JavaScript, running JooUnit tests and packaging the compilation result. Because *Jangaroo* artifacts and dependencies are of type `jar`, transitive dependencies are

automatically added to the compile scope, making it easy to create and maintain modularity throughout your project.

For further information on the *Jangaroo Maven* build process, visit the corresponding documentation in the [Jangaroo Tools Wiki](#).

BOM files

BOM stands for "bill of material" and defines an easy way to manage your dependency versions. The BOM concept depends on the `import` scope introduced with *Maven* 2.0.9, that allows you to merge or include the `dependencyManagement` of a foreign POM artifact in your POMs `dependencyManagement` section without inheriting from it.

```
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>core-bom</artifactId>
  <version>5.4.23</version>
  <type>pom</type>
  <scope>import</scope>
</dependency>
```

The inclusion or merge is done before the actual dependency resolution of your project is done. By the time the actual resolution starts *Maven* does not see any BOM imports but only the merged or included dependencies.

For projects using a framework that provides many artifacts like *CoreMedia* does, this means, that you can fix the versions for all dependencies that are part of that BOM, by simply declaring one dependency.

Of course there are pitfalls when using BOMs and the `import` scope, but the benefits of using BOMs overcome any disadvantages. To prevent you from falling into one of the pitfalls, the following paragraphs will show you how to use the BOM approach correctly.

Chaining BOMs and artifact procurement

Artifact procurement is a feature that some repository management tools like *Nexus* or *Artifactory* offer you to allow your project to use only explicitly configured versions of their dependencies. In addition to the local dependency management in your POM files, artifact procurement is done remotely in your artifact repository. Because of this fact, artifact procurement is much stricter and most commonly only applied in organizations, where securing build infrastructure has the highest priority.

When you chain BOM files, which means that the BOM you import, imports another BOM and so forth, you cannot achieve complete artifact procurement if any POM enforces a different version of a BOM than the version that is used within that chain of its predecessors. This problem stems from the fact that all import scoped dependencies must be resolved in any case, even if your topmost project enforces

a different version. Luckily this only affects POM artifacts, you cannot compile against or which have no effect when deployed to the classpath.

BOM import order

Because the import scope is more likely an `xinclude` on XML basis, ordering of these imports is crucial if the BOMs content is not disjoint, which is most likely the case in presence of chained BOMs.

As a result, it is important to list the BOM imports in reverse order of the BOM import chain. To make sure your update is correct you should therefore always create the effective POM and check the resulting `dependencyManagement` section. To do so execute:

```
$mvn help:effective-pom -Doutput=effective-pom.xml
```

4.1.2 Blueprint Base Modules

CoreMedia Digital Experience Platform 8 introduces a new way of providing default features for *CoreMedia Blueprint*, *Blueprint Base Modules*. Step by step *CoreMedia* will move features from the *Blueprint* workspace to the *Blueprint Base Modules*. All features of the *Blueprint Base Modules* will be described by a public API. The reasons why *CoreMedia* decided to do so are:

- Less source code means faster Maven builds.
- Less source code in *Blueprint* workspace leads to easier migration paths when updating to new versions.

As its name implies, this new module contains *Blueprint* logic and thus depends on the *Blueprint*'s content model. The content model is still part of the *Blueprint* workspace, hence you may customize it. Be aware, that some changes will break the *Blueprint Base Modules*. Read [Section “Content Type Model Dependencies” \[142\]](#) for the list of *Blueprint Base* dependencies to the *Blueprint* content type model.

Read [Section 4.3.1, “Using Blueprint Base Modules” \[141\]](#) for a detailed description of how to develop with the various *Blueprint Base Modules*.



4.1.3 Application Architecture

CoreMedia applications are hierarchically assembled from artifacts:

- Library artifacts are used by
- Component artifacts are used by

- ➔ Application artifacts are used by
- ➔ Packages artifacts

The following sections describe the intention of the given artifact types.

Library Artifacts

Library artifacts contain JAR artifacts with Java classes, resources and Spring bean declarations.

An example is the artifact `cae-base-lib.jar` that contains CAE code as well as the XML files which provide Spring beans.

Component Artifacts

Component artifacts provide a piece of business (or other high level) functionality by bundling a set of services that are defined in library artifacts. Components follow the naming scheme "`<componentKey>-component.jar`". The component artifact `cae-component.jar` for example, bundles all services that are typically required by a CAE web application based project.

Component artifacts are automatically activated on application startup, in contrast to library artifacts. That is, Spring beans and properties are loaded into the application context and servlets and so on will be instantiated. Therefore, you can add a component by simply adding a Maven dependency. No additional steps (such as adding an import to a Spring file) are necessary.

The following files allow you to declare services for a component which are automatically activated:

- ➔ `/META-INF/coremedia/component-<componentname>.xml`:
An entry point for all component Spring beans. Either declared directly or imported from library artifacts.
- ➔ `/META-INF/coremedia/component-<componentname>.properties`:
All configuration options of the component as key/value pairs. These properties might be overridden by the concrete application.
- ➔ `/META-INF/web-fragment.xml`: A Servlet 3.0 fragment of a `web.xml` file that declares component specific servlets, listeners, etc. (see <http://www.oracle.com/technetwork/articles/javaee/javaee6overview-part2-136353.html>)



Consider extending `com.coremedia.springframework.web.ComponentWebApplicationInitializer` instead of writing a `web-fragment.xml` so that your component's servlet configuration (listeners, filters, etc.) can be disabled at application startup time by passing a list of component names as property `components.disabled` to the web application. The property is read using an instance of spring's [Standard Servlet Environment](#).

Application Artifacts

An application artifact is a WAR (web application) file that is ready to be deployed in a servlet container. It consists of one or more component and library artifacts as well as a small layer of code or configuration that glues the components together. Web resources such as image or CSS files might be either directly contained in application artifacts or might be bundled below `/META-INF/resources` inside a shared library or component artifact.

Application artifacts may contain the following files to configure its components:

- `/WEB-INF/web.xml`: Servlet 3.0 web application deployment descriptor, may declare a load order for component web fragments and additional servlet filters, listeners, etc.
- `/WEB-INF/application.xml`: Contains additional Spring configuration which is required by the application and not provided by components.
- `/WEB-INF/application.properties`: Configures components packaged with the application. Values set here override any default application and component configuration in `/WEB-INF/application.xml` and `/META-INF/coremedia/component-<componentname>.properties`.
- `/WEB-INF/logback.xml`: [Logback](#) configuration file for this application. If this file is present, it will override the default log configuration. Any custom log configuration file should include `logging-common.xml`, which defines appenders "file" and "console" and sets the name and location of the log file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="30 seconds">
  <include resource="logging-common.xml"/>
  <!-- define loggers here, for example: -->
  <logger name="com.coremedia" additivity="false" level="info">
    <appender-ref ref="file"/>
  </logger>
  <root level="warn">
    <appender-ref ref="file"/>
  </root>
</configuration>
```

Example 4.2. Including logback-common.xml

```
</root>
</configuration>
```

See [Section “The Logging Component” \[165\]](#) for additional information.

You can specify additional properties files by defining a comma-separated list of paths in a System or JNDI property with the name `propertieslocations`.

Application properties are loaded from the following sources, from the highest to the lowest precedence:

- **System properties:** Useful for overriding a property value on the command-line. The actual property name must be prefixed with "coremedia.application." when set as a system property:

```
$ mvn '-Dcoremedia.application.management.server.remote.url=
      service:jmx:jmxmp://localhost:6666' tomcat7:run
```

- **JNDI context:** Allows deployers to override application properties without modifying the WAR artifact. Object names must be prefixed with "coremedia/application/". For instance, to set the property `management.server.remote.url` via the JNDI context, its value may be added to the application environment in the application's `/WEB-INF/web.xml` or in Tomcat's [context configuration](#):

```
<env-entry>
  <env-entry-name>
    coremedia/application/management.server.remote.url
  </env-entry-name>
  <env-entry-value>
    service:jmx:jmxmp://localhost:6666
  </env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Example 4.3. Setting an environment property in `web.xml`

```
<Context>
  <!-- ... -->

  <Environment
    name="coremedia/application/management.server.remote.url"
    value="service:jmx:jmxmp://localhost:6666"
    type="java.lang.String"
    override="true"/>

  <!-- ... -->
</Context>
```

Example 4.4. Setting an environment property in the context configuration

- **`/WEB-INF/application.properties`:** Build-time configuration of application properties.

- ➔ `/WEB-INF/component-*.properties`: Like `application.properties`, but allows grouping of properties by component.
- ➔ `classpath:/META-INF/coremedia/component-*.properties`: Default component configurations provided by the author of the component. These files should not be modified to configure components for use in a particular application. Instead, their values should be overridden in the application artifact in the files `/WEB-INF/component-*.properties` or `/WEB-INF/application.properties`.

Package Artifacts

A package artifact is a deployable artifact. It may contain one or more application artifacts, which can be tools, web applications or any other resources you need to deploy, for example an Apache configuration overlay or a Tomcat installation. The main purpose of a package artifact is to add an installation infrastructure. In case of simple Zip artifacts this can be a set of shell or batch scripts. In case of a native package, an RPM for example, the native package manager provides the infrastructure to install a package artifact.

Package artifacts may be configured at build time with default values or with filter tokens, that can be replaced after the installation on the target machine (see [Section 4.3.9, “Configure Filtering in the Workspace” \[171\]](#) for details).

Tomcat based Services

In case a package artifact repackages a set of WAR application artifacts, the package artifact wraps a Tomcat configuration stub around one or more web applications and adds the infrastructure to register the Tomcat instance as a service. A Tomcat configuration stub contains only the necessary configuration files of a Tomcat and the web applications it should start. A common Tomcat installation, identified by an environment variable `CATALINA_HOME`, can then be used to start an instance of itself using an environment variable `CATALINA_BASE` pointing to the Tomcat configuration stub. This way you don't need to package a whole Tomcat installation with each service package artifact.

For an overview of the default layout used for Linux/Unix, see [Section 9.3, “Linux / Unix Installation Layout” \[468\]](#).

Redundant Spring Imports

Due to the design of the Spring Framework and the CoreMedia CMS, it is necessary to declare many `<import/>` elements in Spring configuration files, often pointing to the same resource. This slows down the startup of the `ApplicationContext`.

Unfortunately, `org.springframework.beans.factory.xml.XmlBeanDefinitionReader` does not track imported XML files, so redundant `<import/>` elements will lead to Spring parsing the same XML files over and over again (in most

cases, those XML files will contain more `<import/>` elements leading to even more parsing, ...) After moving to Servlet 3.0 resources, for each `<import/>`, the JAR file containing the XML file has to be unpacked. Also, every time that an XML file is completely parsed, Spring reads all Bean declarations, creates new `org.springframework.beans.factory.config.BeanDefinition` instances, overwriting any existing BeanDefinitions for the same bean ID.

This release introduces an optional [Spring Environment](#) property `skip.redundant.spring.imports` that is `true` by default. If set to `true`, the first `<import/>` element will be used and all following, duplicate `<import/>` elements pointing to the same resource will be ignored. The time saved depends on the number of duplicated `<import/>` elements.

Even though this setting is recommended, it may change which bean definitions are loaded. (As explained above, normally, bean definitions may be overwritten by subsequent imports, depending on how `<import/>` elements are used in a web application).



4.1.4 Structure of the Workspace

The *CoreMedia Blueprint* workspace contains the modules, packages, boxes and test-data top level aggregator modules.

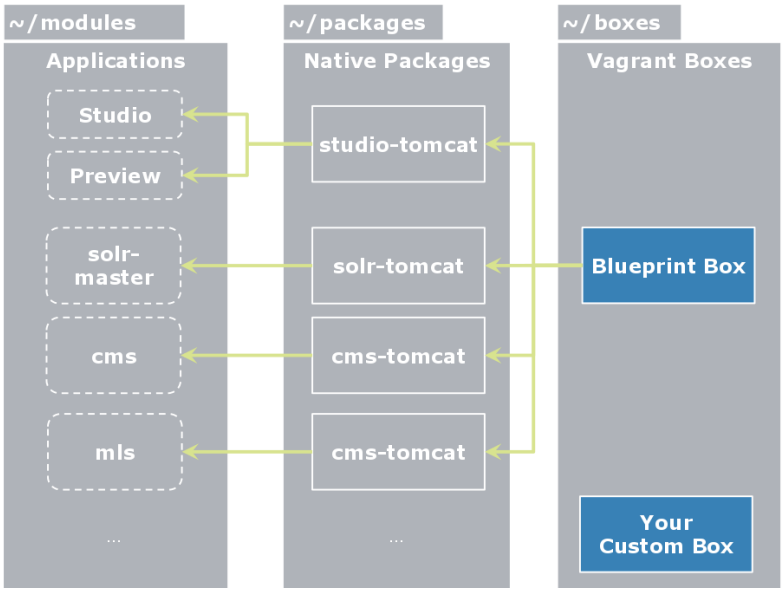


Figure 4.1. Workspace Structure

modules

The `modules` aggregator module is the most important space for project developers. All code, resources, templates and the like is maintained here. You can start all components locally in the `modules` area.

The `modules` hierarchy consists of modules that build libraries and modules that assemble these libraries to applications. Library modules are being built with the standard Maven `jar` packaging type or with the `jangaroo` packaging type, which is a custom packaging type to package *CoreMedia Studio* libraries and plugins.

Most applications created by the modules below the `modules` folder are web applications using the standard Maven `war` packaging type. All other applications are built with the custom `coremedia-application` packaging type. In contrast to `war` modules `coremedia-application` modules are being built with the `coremedia-application-maven-plugin`, a custom plugin tailored to the *CoreMedia .jpf* based application runtime.

The `modules` folder is structured in sub-hierarchies by grouping modules due to their functionality. The main groups are `cae`, `cmd-tools`, `ecommerce`, `editor-components`, `studio`, `search` and `server` which contain the applications defining *CoreMedia DXP 8*.

Beside the application groups you can see a folder named `shared`. Modules belonging to this category cannot be assigned to one group alone but merely provide libraries and APIs for multiple applications. The remaining two groups `extension-config` and `extensions` are required for the new extensions functionality of *CoreMedia Blueprint* workspace.

By default, *CoreMedia Blueprint* workspace ships preconfigured with many extensions such as *Adaptive Personalization* or *Elastic Social*. Many extensions not only touch one application but merely extend many of them. The *CoreMedia Project Extensions* decouples the application from the dependencies it is extended by and lets you centralize and automatically manage the dependencies. See [Section 4.1.5, “Project Extensions” \[114\]](#) for details.

Not all extensions will be used in a project right from the start. In this case, the *CoreMedia Project Extensions* allow you to easily deactivate features that you do not need.

packages

The artifacts, for example WAR and Zip files build by the modules hierarchy are not deployable in a target environment as they contain default configuration values used to simplify development with the workspace. The packages hierarchy adds this configuration flexibility and even more important it adds the integration with the servlet container and the operating systems service infrastructure.

The artifacts build in packages are either Zip or RPM files. For RPM files there is an inbuilt native installation routine whereas the Zip files contain custom installation scripts.

The `packages` hierarchy consists of several sub-hierarchies. The main hierarchies are `services`, `tools` and `apache-overlays`. The `services` sub-hierarchy builds all applications that are built on top of Tomcat as a servlet container. The `tools` sub-hierarchy builds all command-line tools for all services. The `apache-overlays` hierarchy builds overlays to an already installed Apache server, adding only configuration files.

Beside these three hierarchies there is the `package-template` module, that contains all the OS-specific installation code like the scriptlets for the RPM files or the initialization scripts for the Tomcat services.

The `tomcat` hierarchy consists of modules necessary for the Tomcat service infrastructure. The `tomcat-installation` module builds an installable version of a Tomcat distribution. The Tomcat based services share, when installed on the same machine, one installation of Tomcat. They only create instances of Tomcat using the binaries of the distribution together with their own configuration files, which are being provided by the `tomcat-config` artifacts. The `tomcat-server-libs` and `coremedia-tomcat` modules provide class loading extensions.

The `editor-webstart-webapp` module has been placed here as signing and repackaging is closer related to the packages hierarchy as with the modules hierarchy.

boxes

The `boxes` module contains all provisioning code to automatically set up a *CoreMedia Blueprint* system. For local development it also contains a harness in the form of a Vagrant file to startup a virtual machine within VirtualBox. See [Section 4.1.6, “Virtualization and Provisioning” \[119\]](#) for more information.

test-data

The `test-data` folder contains test content to run *CoreMedia Blueprint* with. It can be imported into the content repository by using the *CoreMedia serverimport* tool. Extensions may contain additional test-data folders. For more information have a look at [Section 3.5.3, “Locally Starting the Components” \[85\]](#)

4.1.5 Project Extensions

One of the main goals of *CoreMedia DXP 8* is to offer a developer friendly system with a lot of prefabricated features, that can simply be extended modularly. To this end, CoreMedia provides the Maven based *CoreMedia Blueprint* workspace and the extensions mechanism.

An extension adds new features to one or more CoreMedia components. To preserve a modular structure in the workspace an extension should be developed in its own Maven module and should have submodules for extensions that affect different CoreMedia applications. Assume, for example, that you want to extend your CoreMedia system to integrate external content and render this content with a proof of origin. In this case there is a submodule for the CAE containing new view templates for this content and there is another submodule containing new UI components for *Studio* to search specific sources and create new content.

In *CoreMedia Blueprint* workspace extensions are, in principle, enabled for one component by adding a Maven dependency on the extension module to the component module.

Nevertheless, because extensions often affect more than one component (for example the CAE and *Studio* as shown in [Figure 4.2, “CoreMedia Extensions Overview”](#) [115]) that would mean that a dependency for each affected component has to be declared. If the extension should be disabled, the corresponding dependencies have to be removed from the components. This requires many manual steps and it is error-prone if dependencies are forgotten. The *CoreMedia Blueprint Maven Plugin* is a Maven build extension that does all the dependency management for you. It allows you to define extensions for multiple CoreMedia components in separate modules and to enable or disable them in one place. Find the details described in the following paragraphs.

Extensions

CoreMedia Blueprint
Maven Plugin

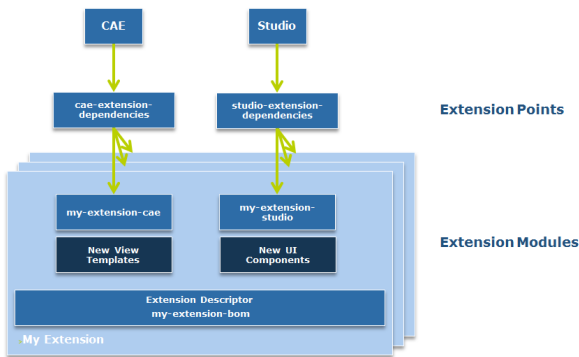


Figure 4.2. CoreMedia
Extensions Overview

How Extensions and Components Integrate

The *CoreMedia Blueprint Maven Plugin* requires the following features in the Maven workspace for operation :

- ➔ Extension Point: Each CoreMedia component that can be extended defines an extension point in the `modules/extensions-config` folder. The extension point is a simple POM file in a component specific submodule. For in-

stance extensions-config/studio-extension-dependencies/pom.xml is the extension point of *CoreMedia Studio*.

- ➔ Extension Descriptor: A simple BOM POM file of the extension that has dependencies on the component specific extension modules. The root POM file of the project has to depend on this POM file in order to activate the extension.
- ➔ coremedia.project.extension.for: A property in the POM files of the component specific extension submodule. The property defines for which CoreMedia component the extension is intended.

The *CoreMedia Blueprint Maven Plugin* is a tool which manages extensions for you. It analyzes the root pom.xml file and its dependencyManagement entries to identify the registered extensions by their extension descriptor. If any are found, each extension module defined in the extension descriptor of the extension is added to the extension point. An application now only needs to depend on its extension points and all necessary dependencies will be added to the applications dependency tree by transitivity. The following sections describe this in detail.

How the CoreMedia Project Maven Extension works

Mapping Components and Extensions

The *CoreMedia Blueprint Maven Plugin* handles the dependencies for you in order to enable or disable an extension. To reach that, a relation between extension module and corresponding component must exist. This relation will be defined by the extension module itself. In other words the extension module exposes its target component. The *CoreMedia Blueprint Maven Plugin* expects a Maven property in the extension module. The name of the property is **coremedia.project.extension.for**. The value of this property describes the target component or more precisely it matches the prefix of one module of the extension points. The *CoreMedia Blueprint Maven Plugin* adds a dependency on the extension module to the matching extension point module (see [Figure 4.3, “Component Mapping” \[116\]](#)).

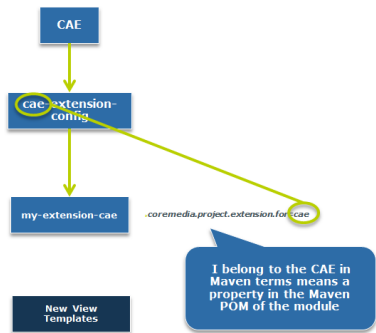


Figure 4.3. Component Mapping

Activating Extensions

As already mentioned the *CoreMedia Blueprint Maven Plugin* allows you to enable and disable an extension in one place. Extensions are enabled by adding the Extension Descriptor to the project's root POM and running the **update-extensions** goal of this Maven plugin. While the Extension Descriptor is a BOM POM you have to import it to the dependencyManagement section of the project's root POM as shown in [Example 4.5, “Enabling an Extension” \[117\]](#). To disable an extension you have to remove the import also followed by the **update-extensions** goal of this plugin. For more information about developing with the *CoreMedia Blueprint Maven Plugin* see [Section 4.3.2, “Developing with Extensions” \[146\]](#).

Example 4.5. Enabling an Extension

```
<dependencyManagement>
  <dependencies>
    ...
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>my-extension-bom</artifactId>
      <version>${project.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>
```

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
    update-extensions
```

The Project Extension mechanism does not corrupt the Maven dependency resolving in any way. The dependencies are added before Maven analyzes dependencies and calculates the build order. Since the managed dependencies are persisted in a standard `pom.xml`, every IDE and other Maven based tool should work as expected.



Example

The example bases on the extension mentioned above; an extension that extends your CoreMedia system to integrate external content and render this content in the CAE. Therefore, it needs sub modules for *Studio* and *CAE* specific parts:

```
--extensions
--externalContent
--externalContent-cae
--externalContent-studio
```

Example 4.6. Module structure of the extension

Each component specific submodule needs to show for which CoreMedia component it is intended. The `pom.xml` file of the `externalContent-studio` module, for example, has to contain the following property:

```
<properties>
  <coremedia.project.extension.for>
    studio
  </coremedia.project.extension.for>
</properties>
```

Example 4.7. Define the component

The extensions need to be defined in the extension descriptor in the BOM POM file of the extension.

```
--extensions
--externalContent
--externalContent-bom
--pom.xml
--externalContent-cae
--externalContent-studio
```

Example 4.8. Module structure with BOM POM

```
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>externalContent-cae</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>externalContent-studio</artifactId>
    <version>${project.version}</version>
  </dependency>
</dependencies>
```

Example 4.9. BOM POM with dependencies on submodules

Now, you have to activate the extension. Simply add a dependency on the BOM POM file to the root `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
```

Example 4.10. Enabling the extension in the root POM file

```
<artifactId>externalContent-bom</artifactId>
<version>${project.version}</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
```

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
    update-extensions
```

The *CoreMedia Blueprint Maven Plugin* will add the dependencies on the `externalContent-cae` and `externalContent-studio` to the respective extension points of *CAE* and *Studio*.

4.1.6 Virtualization and Provisioning

The *CoreMedia Blueprint* workspace supplies a virtualization and provisioning infrastructure to set up a virtual machine (VM) with all services and applications in a fully automated way. To achieve this *Chef* is used as the configuration management tool, a tool that has been proven and has been adopted widely. As a virtualization solution for development purposes, CoreMedia uses *Vagrant* as a wrapper around *Oracle VirtualBox* to configure the virtualized hardware, the network settings and to trigger the provisioning process of *Chef* on the box.

The term provisioning hereby describes the process of converging the state of a box from an arbitrary current state into the desired state configured with *Chef*, whereas the term box describes a virtual machine in the sense of a whole virtualized systems and neither means a *Java Virtual Machine* nor a restriction to *Oracle Virtual Box* as the virtualization solution. The *Chef* setup provided can also be applied to *VMware* images and bare metal systems.

Due to legal restrictions from Oracle, the boxes are configured with OpenJDK. If you want to use the Oracle JDK you have to import the Java community cookbook (<https://supermarket.chef.io/cookbooks/java>) and activate Oracle JDK in your *Vagrant* files.



Boxes Overview

By default, only a simple virtualized environment, where all applications are running on one box, is shipped with the workspace. This box environment provides all the blueprint services and its dependencies necessary for development purposes. For better insight, it provides also a *PSDash* processes dashboard that allows tailing of

logs from within the browser and provides a quick overview to all main system statistics. Visit <http://localhost:8999>.

All links to applications and services can be gathered in the `README.md` at the root directory of the workspace.

Please feel free to add your own custom tailored box scenarios to your workspace to improve your development experience. There are many possible scenarios ranging from a simple persistence box where only database services are running to a box where you test your monitoring stack. With *Vagrant* and *Chef*, you can easily provide reusable test beds for any kind of software.

Vagrant

Vagrant is a tool, that allows you to automate the bootstrapping and configuration of virtualized images and thereby provides you with a workflow to test your code in an isolated environment keeping your developers machine clean from any project specific software. By abstracting the virtualization provider and the configuration provisioner, it is highly adaptable to your projects infrastructure.

By default, CoreMedia uses *Oracle VirtualBox* as the provider and *Chef* as the provisioner, but there are many free plugins adding other providers and provisioners. A list of free plugins can be seen [here](#). Beside these free plugins there are also several commercial plugins, notably the *VMware provider* plugin, which is developed by the Vagrant founder himself.

Vagrants main configuration interface is the *Vagrantfile*, written in a *Ruby DSL* to configure hardware and network characteristics as well as a provisioning process to be triggered after booting the box.

To control the lifecycle of the boxes, *Vagrant* comes with an easy to learn command-line interface. *Vagrant* has commands to start, stop, destroy, suspend and resume the boxes. There are also commands to trigger the provisioning process to update the applications in the boxes. For a complete list of the available commands, either call `vagrant --help` on your shell or visit the official [Vagrant documentation](https://docs.vagrantup.com/v2/) at <https://docs.vagrantup.com/v2/>.

As *Vagrant* uses an approach, where it imports a base box and starts provisioning from that state, CoreMedia provides you with a preconfigured base box, containing only a minimum of preinstalled packages like *Java*. Visit [CoreMedia Vagrant base boxes reference](#) for the current base box state.

Be aware that *Vagrant* is NOT a deployment tool for production use, it is a developer aid to get rid of installing and configuring server software on your laptop. More precisely, it closes the gap between infrastructure and code, allowing you to develop and test complex deployment scenarios from within your workspace.



The Vagrant file

The main file in any *Vagrant* based scenario is the *Vagrantfile* file, it is the *pom.xml* file of the virtualization process. In the workspace you will find it beside the main *pom.xml* at the root directory.

Example 4.11. Vagrantfile Example

```
Vagrant.configure("2") do |config|
  ...
  config.vm.box = "coremedia/base"
  config.vm.box_version = "~> 1.16"
  config.omnibus.chef_version = "12.8.1"
  config.vm.define :blueprint do |blueprint_config|
    blueprint_config.vm.network "private_network", ip:
"192.168.252.100"
    blueprint_config.vm.provider "virtualbox" do |v|
      v.customize ["modifyvm", :id, "--memory", 4096]
      v.customize ["modifyvm", :id, "--cpus", 2]
    end
    blueprint_config.vm.provision "chef_solo" do |chef|
      chef.cookbooks_path = path_cookbooks
      chef.roles_path = path_roles
      chef.add_role("coremedia-box-common")
    end
  end
end
```

In the *Vagrantfile* file, several Ruby blocks describe particular configurations. For example the opening *Vagrant.configure("2") do |config|* block starts the *Vagrant* DSL, whereas the *config.vm.define :blueprint do |blueprint_config|* block starts the configuration of a particular box. Within this configuration block, you will encounter two more blocks. The *blueprint_config.vm.provider "virtualbox" do |v|* block, that defines the *Oracle VirtualBox* provider and the *blueprint_config.vm.provision "chef_solo" do |chef|* block, that starts the *Chef chef-solo* provisioner. For a more comprehensive documentation visit the official [Vagrant documentation](https://docs.vagrantup.com/v2/) at <https://docs.vagrantup.com/v2/>.

Chef

As already mentioned earlier, *Chef* is used as the provisioning tool of choice to fully automate the deployment process of CoreMedia applications. Chef is a widely adopted configuration management framework, that has a steadily growing community and provides free installation instructions (cookbooks) for almost any kind of applications or infrastructures.

To learn more about the *Chef* configuration management tool please refer to the official [Chef documentation](https://docs.chef.io/) at <https://docs.chef.io/>.

Chef Repository

The *CoreMedia Blueprint* workspace provides a Chef infrastructure to manage a complex CoreMedia installation. The parts of the infrastructure are grouped as a

Chef repository, a directory layout convention required to work with *chef-server*, the central management facility for complex deployments. The `chef-repo` directory is located at `boxes/chef/chef-repo` and consists of the following subdirectories:

- `cookbooks`, the logical deployment unit for *Chef* recipes.
- `roles`, a *Chef* construct to bundle recipe executions together with configuration attributes.
- `data_bags` containing data files structured as JSON files.
- `environments` a further topology based grouping construct of *Chef*.

The `chef-repo` directory contains `README.md` files which are part of the official `chef-repo` template. They contain instructions how to use the different constructs.

For the development setup with *Vagrant*, *chef-solo* is currently used as the provisioner, because it is easier to start with. The `chef-repo` though, is prepared to use with *chef-server*. To work with both environments, there are extra recipes, that use the *chef-server* search capabilities to dynamically connect nodes applications by querying nodes by their roles. These recipes have the suffix `_override` and cannot be used together with *Vagrant*.



Cookbooks

The `chef-repo` directory contains all necessary cookbooks to set up a CoreMedia installation. This includes third-party community cookbooks from the official [Chef Supermarket](#) site. The cookbooks found there are being released regularly and CoreMedia strives to update the dependencies in the workspace for every distribution release of *CoreMedia Blueprint*.

In the `boxes/chef/chef-repo/cookbooks` directory you will find cookbooks provided by CoreMedia:

- The `blueprint-yum` cookbook configures the *yum* package management.
- The `coremedia` cookbook handles the deployment and configuration of all CoreMedia applications.
- The `psdash` cookbook installs and configures the [PSDash processes dashboard](#). This cookbook is primary for development purposes, where quick insights into the log files or the basic system statistics are helpful. The dashboard can be accessed at `http://localhost:8999`.

CoreMedia Cookbook

The `coremedia` cookbook is located at `boxes/chef/chef-repo/cookbooks/coremedia` and contains the full spectrum of *Chef* constructs, this includes:

- **recipes** as the installation instructions to be executed by chef.
- **attributes** containing the default configuration attributes for all recipes.
- **resources** (lightweight resources LWR) to compose new recipes from. Currently there are resources for `configuration`, `content`, `probedog` and `workflows`. For a more detailed description view the `boxes/chef/chef-repo/cookbooks/coremedia/README.md` file written in markdown syntax, a common standard for cookbook documentation.
- **providers** (lightweight providers LWP), the implementation of the LWRs.
- **definitions** as a macroized recipe, to reduce the repetitive declaration of *Chef* resources in every recipe.
- **libraries** for concrete ruby implementation of *Chef* APIs.
- **templates** to render files with parameters to disk.

The recipes of the `coremedia` cookbook install either logical units of deployment or configure infrastructure or third-party services.

The recipes installing CoreMedia services are very similar, they all contain one or more `package` resources installing the service and tool RPMs and a `service` resource responsible for the lifecycle of the service. If present, a `probedog` resource watches the availability of the service after it has been (re-)started.

In more complex recipes, the `content_management_server` or the `work_flow_server` recipe, for instance, the additional resources and providers (LWRP) are being used to either import and publish content or upload workflows. The reference about the configuration possibilities of those recipes can also be found in the `README.md` file.

The `configuration` resource is responsible for configuring the CoreMedia services and tools. It represents a *Chef* facade for the configuration properties files located below `/etc/coremedia` on the target machine and triggers the reconfiguration process when necessary.

By default, all *Chef* attributes, together with all information gathered by Chefs inspection tool *ohai* are merged together into one JSON data structure for each Chef node. By defining additional attributes either in roles or environments, you can extend this set. The `configuration` resource accesses all attributes found below the nested hash at `node['coremedia']['configuration']` and compares the ones required for an application with the ones found currently on disk. If both sets don't match, the resource will trigger the reconfiguration of the resource.

All keys listed in the `packages/src/main/filters/default-deployment.properties` file are available for configuration. If, for example, you may want to change the value of the JVM heap size of the *Content Management Server*, you need to set your desired value with the key `configure.CMS_HEAP`. In the

`boxes/chef/vagrant-chef-repo/roles/management.rb`, you can see, how it is done for the vagrant deployment.

Roles

Recipes that should run together on one node are grouped together with attributes into *Chef* roles. In the `boxes/chef/vagrant-chef-repo/roles` directory, you find roles configured only for the *Vagrant* development setup.

4.2 Administration and Operation

This chapter describes how to administrate and operate features of *CoreMedia Digital Experience Platform 8* and how to release, deploy and maintain a *CoreMedia DXP 8* System.

- [Section 4.2.1, “Performing a Release” \[125\]](#) gives a brief introduction on how to perform a *Maven* release and how to upload the deployment artifacts correctly.
- [Section 4.2.2, “Deploying a System” \[127\]](#) describes how to deploy the uploaded artifacts.
- [Section 4.2.3, “Upgrade a System” \[137\]](#) describes how you can upgrade a system.
- [Section 4.2.4, “Rollback a System” \[139\]](#) describes how you can rollback deployed packages.
- [Section 4.2.5, “Troubleshooting” \[140\]](#) gives some hints in case of problems.

4.2.1 Performing a Release

This section describes how you can release the workspace with *Maven* and upload the artifacts to a repository. Before you continue, make sure you have configured the `scm` and `distributionManagement` section as described in the [Section 3.3.3, “Configuring the Workspace” \[46\]](#).

At the heart of every release process is the `maven-release-plugin`, which divides the process into two phases, the `prepare` and the `perform` phase.

In the `prepare` phase *Maven* sets a release version, builds the workspace and if that succeeds, it commits and tags the state. In a second step the version is bumped up to the next `SNAPSHOT` version.

During the `perform` phase the release committed tag is being checked out to the `target/checkout` directory, from where it is being build. During this `perform` build, the release artifacts are being uploaded to a *Maven* repository.

The default call you need to execute is therefore:

```
$ cd $CM_BLUEPRINT_HOME
$ mvn release:prepare release:perform
```

Upload Libraries, Components and Applications

During a release, you only need to upload the artifacts of the modules below the `$CM8_BLUEPRINT_HOME/modules` folder into a *Maven* repository as there is no need to reuse the artifacts below the `$CM8_BLUEPRINT_HOME/packages` folder

in other projects, they are for deployment purposes only. Because of this, the `maven-deploy-plugin` is configured to skip the `deploy` goal for these modules.

Upload Packages

Because there are Maven repository servers with the capability to serve a Maven repository with artifacts of type `rpm` as a Yum repository, you will find a hook to the `$CM8_BLUEPRINT_HOME/boxes` module to upload the `rpm` artifacts to a dedicated repository. Currently Nexus and Artifactory are supporting Yum repositories.

To deploy RPM packages, you need to activate the Maven profile `repository-upload`.

By default, the profile can be configured using the properties shown in the table below:

property	usage
<code>rpm.upload.repository.url</code>	The URL to upload the artifacts to. Defaults to <code>\${project.distributionManagement.repository.url}</code>
<code>rpm.upload.repository.id</code>	The id of the server authentication in the <code>settings.xml</code> . Defaults to <code>\${project.distributionManagement.repository.id}</code>

Table 4.1. RPM deployment properties

Because repository servers like Nexus prohibit the deployment of snapshot artifacts into release repository and vice versa, you need to overwrite the repository URL to upload snapshot RPM artifacts. In case of a CI build you can then use a profile defined in the `settings.xml` file to overwrite the properties. In most cases there is already such a snapshot profile to add snapshot repositories and you can reuse it. In any case, the profile must include the properties, for example :

```
<profile>
  <rpm.upload.repository.url>REPO_URL</rpm.upload.repository.url>
  <rpm.upload.repository.id>REPO_ID</rpm.upload.repository.id>
</profile>
```

Example 4.12. Snapshot Profile

To activate the hook during the Maven release, you need to add the profile to the comma separated list of profiles defined in the `releaseProfiles` configuration property of the `maven-release-plugin`. You can find the configuration in the `pluginManagement` section of the `$CM7_BLUEPRINT_HOME/pom.xml`.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>Configuration of maven-release-plugin</artifactId>
```

Example 4.13. maven-release-plugin

```
<version>2.3.2</version>
<configuration>
  <autoVersionSubmodules>true</autoVersionSubmodules>
  <pushChanges>false</pushChanges>
  <localCheckout>true</localCheckout>
  <preparationGoals>clean verify</preparationGoals>
  <releaseProfiles>
    postconfigure,
    repository-upload
  </releaseProfiles>
  <arguments>-Ppostconfigure</arguments>
</configuration>
</plugin>
```

If you want to upload the RPM artifacts manually, you need to execute the `deploy` goal together with the profile, for example:

```
$ cd $CM7_BLUEPRINT_HOME/boxes
$ mvn deploy -Prepository-upload
```

Upload RPM Archive to S3

If you want to deploy the system within *Amazon EC2* there is another hook to create an archive containing all RPM files and upload it to *Amazon S3*. This hook can be activated with the Maven profile `s3-upload`, defined in the `$CM7_BLUEPRINT_HOME/boxes/pom.xml`.

Upload Zip Packages

As there is no standardized deployment process for applications packaged as Zip files, there is no preconfigured solution within the workspace as well. If you still want to upload the Zip archives directly to their target machines, you can use the `wagon-maven-plugin` to achieve this. You will find the plugin's documentation [here](#).

4.2.2 Deploying a System

After artifacts have been released or uploaded to a repository, you need to set up a system by deploying those artifacts. Currently, CoreMedia offers you three possible deployment strategies:

- ➔ The automatic provisioning process using *Chef*.
- ➔ The semiautomated processes using `rpm` and `yum`.
- ➔ The semiautomated process using the proprietary install scripts of the Zip archives.

Deployment with Chef

With the *CoreMedia Blueprint* workspace, CoreMedia provides you not only with RPMs but also with a prefabricated automated way to set up a whole system from scratch using *Chef* as a provisioning tool.

The whole idea of using *Chef* is to fully automate the installation and upgrade process and integrate its configuration *infrastructure as code* into the workspace. Bundled together, installation configuration and project code, you can reduce the risk of not being able to set up a system specific state from scratch after a breakdown.

With *Chef*, you do not have to install and configure all services manually on each machine by yourself, but *Chef* will take care of it. This includes not only the installation of *CoreMedia Digital Experience Platform 8* services, but also third-party software like *MongoDB* and *MySQL*.

Using Chef-Solo

If you start with *Chef* and simply want to set up a system on a single machine, you should use the `chef-solo` client utility as it doesn't require a *Chef Server* or an account for *Hosted Chef*.

Prerequisites

Before you can start deploying the components with `chef-solo`, you need to fulfill the following requirements:

- ➔ Install *chef-solo*.
- ➔ Provide the RPM artifacts either from a local directory or from a remote YUM repository.

Additionally, there are some optional requirements, mostly used to set up a development or test system:

- ➔ Provide a Zip archive of a content export, either as a local file or from a remote accessible URL.

Installing Chef Client

To install *Chef Client* (including `chef-solo`) execute the following command:

```
sudo true && curl -L https://www.chef.io/chef/install.sh | sudo bash
```


Configuring the `solo.rb` file

The file to configure the `chef-solo` client is `solo.rb`, it contains the paths' to the cookbooks and the roles and the logging configuration of *Chef*. You have to create this file. A simple `solo.rb` file should look like the example below.

Example 4.14. An example `solo.rb` file

```
# CHEF ENVIRONMENT CONFIGURATION
log_level      :info
log_location   "/var/log/chef/chef-solo.log"
cookbook_path  "PATH_TO_YOUR_CHEF_REPO/cookbooks"
role_path      "PATH_TO_YOUR_CHEF_REPO/roles"
```

Install *Berkshelf* and vendor the cookbooks. To do so,

```
cd boxes/chef/chef-repo
berks vendor
```

Afterwards the new generated `berks-cookbooks` directory must be added to the `cookbooks` path of the `solo.rb`.

Configuring the `node.json` file

The `node.json` file contains all the configurations you want to apply with the `chef-solo` run. This includes the attributes you want to set but also the roles and recipes you want to apply. The example below configures a complete *CoreMedia 7* run.

Example 4.15. An example `node.json` file

```
{
  "fqdn": "YOUR_ALIAS",
  "mysql": {
    "allow_remote_root": true,
    "bind_address": "0.0.0.0",
    "use_upstart": false,
    "server_repl_password": "coremedia",
    "server_debian_password": "coremedia",
    "server_root_password": "coremedia",
    "tunable": {
      "wait_timeout": "7200"
    },
    "client": {
      "packages": ["mysql-community-client", "mysql-community-devel"]
    },
    "server": {
      "packages": ["mysql-community-server"]
    }
  },
  "coremedia": {
    "db": {
      "schemas": ["cm7management",
                  "cm7master",
                  "cm7replication",
```

```

        "cm7cafeeder",
        "cm7mcafeeder"]
    },
    "yum":{
      "local": {
        "path": "YOUR_LOCAL_RPM_REPO",
        "archive": "YOUR_RPM_REPO_ARCHIVE_URL"
      }
    },
    "content_archive": "YOUR_CONTENT_ARCHIVE"
  },
  "configuration":{
    "configure.STUDIO_TLD":"YOUR_ALIAS",
    "configure.DELIVERY_TLD":"YOUR_ALIAS",
    "configure.CROWD_APP_NAME":"YOUR_CROWD_APP_NAME",
    "configure.CROWD_PASSWORD":"YOUR_CROWD_PASSWORD",
    "configure.CROWD_SERVER":"YOUR_CROWD_SERVER:8443",
    "configure.ELASTIC_MAIL_HOST":"YOUR_MAIL_SERVER",
    "configure.DELIVERY_REPOSITORY_HTTP_PORT":"42080",
    "configure.DELIVERY_SOLR_PORT":"44080"
  },
  "logging":{
    "default":{
      "com.coremedia":{"level":"info"},
      "cap.server":{"level":"info"},
      "hox.corem.server":{"level":"info"},
      "workflow.server":{"level":"info"}
    }
  },
  "tomcat":{
    "manager":{
      "credentials":{
        "admin":{
          "username":"admin",
          "password" : "tomcat",
          "roles"      : "manager-gui"
        },
        "script": {
          "username" : "script",
          "password" : "tomcat",
          "roles"      : "manager-jmx,manager-script"
        }
      }
    }
  },
  "run_list":[
    "recipe[blueprint-yum::default]",
    "recipe[coremedia::chef_logging]",
    "recipe[coremedia::reporting]",
    "recipe[mysql::server]",
    "recipe[coremedia::db_schemas]",
    "recipe[mongodb::default]",
    "recipe[coremedia::solr_master]",
    "recipe[coremedia::master_live_server]",
    "recipe[coremedia::content_management_server]",
    "recipe[coremedia::workflow_server]",
    "recipe[coremedia::cafeeder_preview]",
    "recipe[coremedia::cafeeder_live]",
    "recipe[coremedia::studio]",
    "recipe[coremedia::studio_apache]",
    "recipe[coremedia::certificate_generator]",
    "recipe[coremedia::delivery]",
    "recipe[coremedia::delivery_apache]"
  ]
}

```

Before you can execute `chef-solo` with this configuration, you need to replace all uppercase values starting with `YOUR_`. The following table lists the most important ones:

Table 4.2. *node.js* configurations

Token	Replacement
YOUR_ALIAS	If your machine has an alias and you want the system to use it instead of the resolved hostname, you can set it here.
YOUR_LOCAL_RPM_REPO	The path of the local directory containing the RPM artifacts. If you want to use this option, remove the attribute <code>coremedia['yum']['local']['archive']</code> .
YOUR_LOCAL_RPM_REPO_ARCHIVE_URL	The URL to a remote archive of the RPM artifacts. If you want to use this option, you can remove the attribute <code>coremedia['yum']['local']['path']</code> and <i>Chef</i> will use the default for this value (<code>/shared/rpm-repo</code>).
YOUR_LOCAL_CONTENT_ARCHIVE	An array of paths to local Zip archives containing content, typically an archive with global content and some content archives of extensions which you have activated.

Now, you can simply start *chef-solo* by executing:

```
chef-solo -c solo.rb -j node.json
```

Configuring a Jenkins Setup

As described above, RPM repository and example content can be specified with and retrieved from a remote HTTP URL. For a quick retest system with *Jenkins*, you can use the REST API of *Jenkins* to retrieve those archives via the `lastSuccessfulBuild` URLs. All you need to do is activate archiving for the `boxes/target/shared/rpm-repo/` folder and if you have a corresponding content job, you can apply this pattern there too.

For the *chef-repo*, you can choose between using the archived artifacts or directly reference the jobs workspace using the `*zip*` controller of *Jenkins*. The *Jenkins* job then only needs execute a script, that downloads and extract the *chef-repo* archive and configure the paths in the `solo.rb` correctly.

The script itself can be committed together with the `node.json` in the VCS that the *Jenkins* job checks out.

The following list shows example URLs using `lastSuccessful` artifact URLs for RPM repository and content and a direct workspace URL for the chef-repo:

- ➔ http://MY_JENKINS/job/MY_CM7_JOB/lastSuccessfulBuild/artifact/boxes/target/shared/rpm-repo/*zip*/rpm-repo.zip
- ➔ http://MY_JENKINS/job/MY_CM7_CONTENT_JOB/lastSuccessfulBuild/artifact/target/content-users.zip
- ➔ http://MY_JENKINS/job/MY_CM7_JOB/ws/boxes/chef/chef-repo/*zip*/chef-repo.zip

Using PostgreSQL with Chef

The following steps show how to modify the deployment to use a PostgreSQL database instead of MySQL. The Chef repository in the workspace is prepared for a PostgreSQL deployment, but the included roles must be changed.

1. In the base role, set the `coremedia.db.type` attribute to `postgresql` so that the PostgreSQL schema creation scripts are used. Depending on your operating system you may need to change several attributes of the PostgreSQL cookbook.

- ➔ For CentOS 6 installations, more attributes need to be adjusted.

```
name "base"
description "The base role for CoreMedia nodes"

override_attributes {
  "java" => {"jdk version" => "7"},
  "coremedia" => {"db" => {"type" =>
    "postgresql"}},
  "coremedia",
    "mongodb" => {"cluster_name" =>
    "postgresql" => {
      "version" => "9.2",
      "dir" =>
"/var/lib/pgsql/9.2/data",
      "server" => {
        "service_name" =>
"postgresql-9.2",
        "packages" =>
["postgresql92-server"]
      },
      "client" => {"packages" =>
["postgresql92-devel"]},
      "enable_pgdg_yum" => true,
      "password" => {"postgres" =>
"coremedia"}
    }
  }

run_list "recipe[blueprint-yum::default]",
  "recipe[java]"
```

*Ex-
ample 4.16. base.rb
for CentOS 6*

2. In the management and replication roles, replace the `mysql::server` recipe by the `postgresql::server` recipe. Adapt the `coremedia.configuration` attributes.

Example 4.17. management.rb

```

name "management"
description "The role for CoreMedia Management nodes"

override_attributes "coremedia" => {
  "db" => {"schemas" => %w(cm7management cm7master cm7cafeeder
cm7mcafeeder)},
  "configuration" => {
    "configure.CMS_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.CMS_DB_DRIVER" => "org.postgresql.Driver",
    "configure.MLS_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.MLS_DB_DRIVER" => "org.postgresql.Driver",
    "configure.WFS_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.WFS_DB_DRIVER" => "org.postgresql.Driver",
    "configure.CAEFEEDER_PREVIEW_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.CAEFEEDER_PREVIEW_DB_DRIVER" =>
"org.postgresql.Driver",
    "configure.CAEFEEDER_LIVE_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.CAEFEEDER_LIVE_DB_DRIVER" => "org.postgresql.Driver"
  }
}

run_list "role[base]",
  "recipe[postgresql::server]",
  "recipe[coremedia::db_schemas]",
  "recipe[mongodb]",
  "recipe[coremedia::solr_master]",
  "recipe[coremedia::master_live_server]",
  "recipe[coremedia::content_management_server]",
  "recipe[coremedia::workflow_server]",
  "recipe[coremedia::cafeeder_preview]",
  "recipe[coremedia::cafeeder_live]"

```

Example 4.18. replication.rb

```

name "replication"
description "The role for CoreMedia Replication nodes"

override_attributes "coremedia" => {
  "db" => {"schemas" => %w(cm7replication)},
  "configuration" => {
    "configure.RLS_DB_URL" =>
"jdbc:postgresql://localhost:5432/coremedia",
    "configure.RLS_DB_DRIVER" => "org.postgresql.Driver",
  }
}

run_list "role[base]",
  "recipe[coremedia::management_configuration_override]",
  "recipe[postgresql::server]",
  "recipe[coremedia::db_schemas]",
  "recipe[coremedia::solr_slave]",
  "recipe[coremedia::replication_live_server]"

```

Deployment with Yum

This section covers the deployment process of the *CoreMedia* services using the `rpm` and `yum` utilities directly. It will not cover how to install and configure third-party RPM files such as *MongoDB* or *MySQL*.

Prerequisites

In order to proceed you need to make sure that you successfully meet the following prerequisites:

- A running *MongoDB* instance or cluster, either on the same machine or somewhere else.
- A running *MySQL* instance, either on the same machine or somewhere else.
- A configured *YUM* environment, the `yum` client must be on the `PATH`.
- Either the package `createrepo` must be installed, that is `createrepo` must be on the path or the RPM files must be accessible on a remote YUM repository. To install `createrepo`, call `sudo yum install createrepo`.

Creating a YUM repository

If you want to install the RPM files from a local RPM repository, make sure that you copied all RPM files to a directory and call `createrepo` on that path. If you are using a remote repository, you can skip this step.

Now you need to register a repository configuration in your `/etc/yum.repos.d/`. To do so create a file below that directory, for example `/etc/yum.repos.d/cm8-repo`, and add the following content.

```
[cm8]
name=CoreMedia 8 Repo
baseurl=file://PATH_TO_YOUR_REPO
gpgcheck=0
enable=1
```

Example 4.19. YUM repository

If you are using a remote repository you need to specify the `baseurl` property accordingly.

Installing the RPMs

To install the RPMs, use the `yum` utility to install all selected packages at once, for example:

```
sudo yum install cm8-*
```

for all matching packages or specify a list of packages by calling

```
sudo yum install [PACKAGE_1] ... [PACKAGE_N]
```

Similar to the roles in Chef that group recipes such as packages to apply to your node, you can create groups within your YUM repository. This way, if there was a group `bluepring-delivery`, you could simply use

```
sudo yum groupinstall "blueprint-delivery"
```

to install a set of RPM files at once, without knowing the concrete packages included. To create groups use the `yum-groups-manager` utility.

After you have installed all RPM files and in case you are using the post-configuration approach, you need to adapt the default configuration defined in the property files below `/etc/coremedia` and reconfigure all services and tools. To skip this step you may provide the correct configuration files in advance. As a start you can use the default configuration files generated by the build and adapt them to your needs. You find these files below the `boxes/target/shared/configuration-templates` folder in your workspace.

To reconfigure the services and tools, you need to call `sudo service [SERVICE NAME] reconfigure` for each service and `/opt/coremedia/[TOOL_NAME]/reconfigure-tool.sh` for each tool.

Starting the Services

To start the services simply refer to the standard `service` utility. By default, all Tomcat based services created with the *Blueprint* workspace offer the following methods:

- ➔ `start` - to start the service
- ➔ `stop` - to stop the service
- ➔ `restart` - to restart the service
- ➔ `status` - to check the status of the service
- ➔ `reconfigure` - to reconfigure the service
- ➔ `reload` - to reconfigure and restart the service

The status method does not reflect the status of any of the contained web applications, it merely checks the presence of a process with a matching process id.



The order in which the services need to be started depends on many factors including customizations done in a project. Most CoreMedia applications can be started independently of each other but a safe order would look like the following:

- `service cm7-mls-tomcat start` (Master Live Server)
- `service cm7-cms-tomcat start` (Content Management Server)
- `service cm7-wfs-tomcat start` (Workflow Server)
- `service cm7-solr-master-tomcat start` (Solr Search Engine)
- `service cm7-cafeeder-preview-tomcat start` (Preview CAE Feeder)
- `service cm7-cafeeder-live-tomcat start` (Live CAE Feeder)
- `service cm7-studio-tomcat start` (Studio + Preview, WebDAV and Site Manager)
- `service cm7-delivery-tomcat start` (Delivery CAE)

To reload *Apache*, call **service httpd restart**

Deployment with Zip (Linux)

If you cannot apply the RPM files because RPMs are not supported by your target platform, you can use the Zip files built aside the RPMs. The Zip files contain `install.sh` and `uninstall.sh` scripts to install the applications and services. Even if these scripts internally call the scriptlets used in the RPM approach, they won't provide you with an upgrade mechanism like the RPM approach. That is why it is highly recommended to use the RPMs if possible.

To install a service or a tool with the Zip files you need to do the following:

Installing a service

1. Extract the Zip file to an arbitrary folder except any folder below `/opt/coremedia` as this may lead to conflicts.
2. For the first installation, you may edit the property file below the `INSTALL` folder. It is the one used for post-configuring the first installation. Alternatively you skip this step and reconfigure the application after the installation as described later on.
3. Execute the `install.sh` script.
4. After the installation script has succeeded configure the applications property file below `/etc/coremedia` and reconfigure the service or tool. A service can be reconfigured calling **service SERVICE_NAME reconfigure** whereas the tools provide you with a `reconfigure-tool.sh` script below the installation directory of the tool, such as `/opt/coremedia/cm8-cms-tools` for the *Content Management Server* tools.

Now you have installed *CoreMedia DXP 8* applications on Linux using the Zip files. To start the services follow the instructions shown in [section “Starting the Services” \[135\]](#).

To uninstall a service or a tool you simply need to execute the `uninstall.sh` script in the specific installation directory.

Uninstall service

4.2.3 Upgrade a System

This section describes, how you upgrade a system using the deployment options provided by *CoreMedia DXP 8*.

Upgrading with Chef

Upgrading a system that is managed with *Chef* is as simple as deploying the system with *Chef*. This stems from the fact, that *Chef* guarantees idempotency of its runs which means that it converges the *Chef* resources of a node from an arbitrary state into the desired state. Because this is only guaranteed on the level of *Chefs* domain specific objects, the recipes containing the resources must implement the idempotency in their resources.

The `coremedia` cookbook implements idempotence for its recipes concerning the versions of the installed packages and their configuration. This means, that you only need to configure the versions you want to install and *Chef* will install them on the next run. Due to the implementation of *Chefs* package resource, there are two different upgrade strategies:

- ➔ upgrade to the latest available version, which is default behavior.
- ➔ upgrade to a specific version.

If you want to upgrade all packages to a specific version, the attribute you need to configure in your roles is `coremedia['version']['global']`; if you want to upgrade only specific packages to a version, you can add an attribute, mapping the package name to the version aside the `global` attribute. The example below shows such a configuration:

```
...
override_attributes "coremedia" => {
  "version" => {
    "global" => "14-1.release",
    "cm7-cms-tomcat" => "14-2.release"
  }
}
```

Example 4.20. upgrading to a specific version

When *Chef* converges the node, the recipes of the `coremedia` cookbook follow a standard process to upgrade the affected services.

1. Install the new packages
2. Restart the services
3. Verify the successful restart by probing the service with a *CoreMedia Probedog*

The most important part of this process is the fact, that all service restarts will be scheduled together to the end of the *Chef* run. This way, a service will only be restarted if all packages could be successfully installed.

Upgrade with Yum

The first step, you should do, when you want to update your system, is to find out what package updates are available. With YUM, you simply call `yum list updates`

```
cm8-*
```

In a second step, you may check if your configuration files below `/etc/coremedia` are still up to date or if there are any changes required. You can do that manually by inspecting the package description for each update or you can update the packages and see if there are missing tokens. By default, the CoreMedia RPM files will check if all necessary properties are defined in the configuration file, if that already exists. If properties are missing, the package won't be installed and the missing properties will be reported on the console or in the system log at `/var/log/messages`.

To print out the description of an RPM, you need to execute:

```
yum info <PACKAGE NAME>
```

If for example, you execute `yum info cm8-wfs-tomcat` the RPM metadata will be printed and in the description, you will see a list of properties you need to define.

```
Name       : cm8-wfs-tomcat
Arch       : noarch
Version    : 16
Release    : 1.develop.1362133163
Size       : 17 M
Repo       : installed
From repo  : local
Description: This rpm packages cm7-wfs-tomcat.
           : revision: 20130301-1119
           : To configure this service you have to define a
           : configuration property file at
           : /etc/coremedia/cm7-wfs-tomcat.properties containing:
           : configure.CMS_DB_URL configure.CMS_HOST
           : configure.WFS_IP configure.WFS_HOST
           : configure.WFS_HEAP configure.WFS_PERM
           : configure.CMS_DB_USER configure.CMS_DB_PASSWORD
```

Example 4.21. Yum info

Given that your YUM configuration is correct and all packages are accessible from your YUM repositories, you have to manually proceed the following steps to upgrade your services:

1. `sudo yum update cm8-*` for all *CoreMedia DXP 8* packages. It is important, that you upgrade all packages within one transaction, as it makes it easier to roll back, if something went wrong.
2. After all packages have been installed successfully, you need to restart all CoreMedia services in the same order you started them during the initial deployment.

4.2.4 Rollback a System

This section describes, how you can rollback deployed *CoreMedia Digital Experience Platform 8* packages using the means, provided with the project package structure.

Rollback with Chef

Chef does not offer a specific rollback procedure, but in a proper Chef setup, you have all infrastructure as code within your cookbooks and roles and together with the idempotency *Chef* guarantees, rolling back a release should only require you to rollback your *Chef* configuration into the desired state and wait until *Chef* finished the converging of your nodes.

In case you simply want to reinstall an older version of the *CoreMedia DXP 8* packages, all you need to do is to set the specific version you need to rollback to. See [Section 4.2.3, “Upgrade a System” \[137\]](#) for a description how to set specific versions.

Rollback with YUM

To rollback the packages of a former installation you can undo the YUM transaction that made the upgrade:

1. Enable the rollback feature in YUM by setting `tsflags=repackage` in the `/etc/yum.conf`.
2. Find out the transaction ID of the upgrade you want to revert. `sudo yum history` will list all the transactions. With `sudo yum history info [transaction id]` you can show details of a transaction.
3. Roll back the transaction by calling `sudo yum history undo [transaction id]`.

If you want to downgrade one or all packages to a specific version you need to use the `yum downgrade` command together with the full qualified package names you want to downgrade, for example:

```
sudo yum downgrade cm8-cms-tomcat-14-1.release
```

Rollback with rpm

1. Enable the rollback feature by setting `%_repackage_all_erasures 1` in the `/etc/rpm/macros`.
2. Rollback to a specific date by executing

```
rpm -Uhv --rollback '<DATE>'
```

The date/time can be of the form `9:00 am` or `4 hours ago` or `december 25`.

4.2.5 Troubleshooting

Chef run fails when trying to install MongoDB

Possible cause:

External MongoDB Yum repository is not available.

Solution:

Please contact the support for a mirrored repository URL that can be set up in the Chef attributes.

Cannot access the CAE behind an Apache server on Red Hat Enterprise Linux

Possible cause:

The SELinux policies are not set correctly

Solution:

Make sure that `httpd_can_network_connect` is set to "1" via:

```
setsebool -P httpd_can_network_connect 1
```

You are running all components on a single host and get `OutOfMemory` exceptions in all CoreMedia components with the error message "unable to create new native Thread", most often during the creation of RMI connections

Possible cause:

The process limit of your operation system for the "coremedia" user is too low.

Solution:

Monitor the process limit and, if necessary, increase it or deploy your CoreMedia system on more than one host.

4.3 Development

This chapter describes how you can customize your CoreMedia system in the *CoreMedia Blueprint* workspace. However, it does not describe how you, for example, write a *Studio* plugin or a CAE template; this is explained in the component's specific manual. Instead, it describes how you can use the workspace mechanisms to include your extensions and where you can add your own code or configuration.

- [Section 4.3.2, “Developing with Extensions” \[146\]](#) describes how you can add and remove extensions using the *CoreMedia Project Maven Build Extension*. The extensions mechanism is explained in detail in [Section 4.1.5, “Project Extensions” \[114\]](#).
- [Section 4.3.3, “Extending Content Types” \[155\]](#) describes how you can add your own content types. You will find more details on content types in the [Content Server Manual].
- [Section 4.3.4, “Developing with Studio” \[157\]](#) describes how you can add Studio modules to the list of studio plugins.
- [Section 4.3.5, “Developing with the CAE” \[161\]](#) describes how you can add extensions to the CAE and how you run performance tests.
- [Section 4.3.7, “Adding Common Infrastructure Components” \[165\]](#) describes how you can add the default structure components for logging and JMX to your own web applications.
- [Section 4.3.8, “Managing Properties in the Workspace” \[170\]](#) describes how you can add properties to new or existing components and how you assign values to these properties.
- [Section 4.3.9, “Configure Filtering in the Workspace” \[171\]](#) describes how you can filter the configuration during deployment.

4.3.1 Using Blueprint Base Modules

This section describes how the *Blueprint Base Modules* are integrated into *CoreMedia Blueprint* and how a developer might customize and configure all the various modules or even replace certain modules completely.

CoreMedia Blueprint uses *Blueprint Base Modules* as binary Maven dependencies but CoreMedia provides access to the source code via Maven source code artifacts. IDE's like JetBrains IntelliJ Idea are able to download those sources automatically for a certain class by evaluating its correspondent Maven POM file.



Content Type Model Dependencies

As its name implies, the *Blueprint Base Modules* contain Blueprint logic and thus depend on the Blueprint's content type model. The content type model is still part of the Blueprint workspace, hence you may customize it. Be aware, that changes might affect or even break the *Blueprint Base Modules*. The following table shows an overview of the content types which are relevant for the *Blueprint Base Modules*. Details are explained in the sections about the particular modules.

Content Type (Properties)	Module
CMLinkable(localSettings, linkedSettings)	Settings
CMTeaser(target)	Settings
CMNavigation	Settings
CMSettings	Settings

Table 4.3. Content type model dependencies

The Settings Service

Settings are a flexible way to enable editors to configure application behavior via content changes within *CoreMedia Studio* without the need to redeploy a web application. *CoreMedia Blueprint* uses the `com.coremedia.blueprint.base.settings.SettingsService` to read certain settings from various different sources. This section describes how you can use the settings service in your own projects.

Read [Section 6.3.3, “Settings” \[268\]](#) for a description of why you want to use settings and how to do it from an editors perspective.



The setting* Methods

```
public interface SettingsService {
    <T> T setting(
        String name,
        Class<T> expectedType,
        Object... beans);

    <T> T settingWithDefault([...]);

    <T> List<T> settingAsList([...]);

    <K, V> Map<K, V> settingAsMap([...]);

    [...]
}
```

All `setting*` methods are actually just variants of the basic `setting` method. Some provide additional convenience like `settingWithDefault`, others have complex return types which cannot be expressed as a simple type parameter, for example `settingAsList`. All `setting*` methods have some common parameters which are described in [Table 4.4, “Parameters of the settings* methods” \[143\]](#). For detailed descriptions of the `setting*` methods please consult the API documentation of the `SettingsService`.

Table 4.4. Parameters of the settings* methods

Parameter	Description
<code>name</code>	The name (or key) of the setting to fetch.
<code>expectedType</code>	The type of the returned object. This parameter allows for type safety and prevents you from unchecked casts of the result. For the <code>settingAsList</code> method, the <code>expectedType</code> parameter determines the type of the list entries, not the list itself. <code>settingAsMap</code> has separate type arguments for keys and values of the result map.
<code>beans</code>	Settings are always fetched for one or multiple targets, which are passed by the <code>beans</code> vararg parameter. In the <i>Blueprint</i> 's default configuration the <code>SettingsService</code> supports content objects, content beans, pages, sites and some other kinds of beans.

Configuring the Default Settings Service via SettingsFinders

The *Blueprint Base Modules* not only defines the interface of how to evaluate settings but also provides an implementation and a Spring bean.

```
<beans>
  <bean id="settingsService"
        class="c.c.b.base.settings.impl.SettingsServiceImpl">
    <property name="settingsFinders" ref="settingsFinders"/>
  </bean>

  <util:map id="settingsFinders">
  </util:map>
</beans>
```

The plain `SettingsService` has no lookup logic for settings at all, but it must be configured with `SettingsFinders`. A `SettingsFinder` implements a strategy how to determine settings of a particular type of bean. *CoreMedia DXP 8* provides some preconfigured `SettingsFinders` for popular beans like content objects. It can be modified and enhanced with custom `SettingsFinders` for arbitrary bean types. As you can see, the default settings service only needs one property, which is a map named `settingsFinders`. The keys of that map must be fully qualified Java class names and its values are references to concrete `SettingsFinder` beans.

Example 4.22. The Spring Bean Definition for the Map of Settings Finder

```
<util:map id="settingsFinders">
  <entry key="com.coremedia.cap.content.Content"
    value-ref="cmlinkableSettingsFinder"/>
  <entry key="com.coremedia.cap.multisite.Site"
    value-ref="siteSettingsFinder"/>
</util:map>

<bean id="cmlinkableSettingsFinder"
  class="c.c.b.base.settings.CMLinkableSettingsFinder">
  <property name="cache" ref="cache"/>
  <property name="hierarchy" ref="navigationTreeRelation"/>
</bean>

<bean id="siteSettingsFinder"
  class="c.c.b.base.settings.SiteSettingsFinder"/>
```

The example above shows a map with two settings finders. One is supposed to be used for target beans of type `com.coremedia.cap.content.Content` and the other for targets of type `com.coremedia.cap.multisite.Site`.

In order to determine the appropriate settings finder for a given target bean the settings service calculates the most specific classes among the keys of the settings finders map which match the target bean. For the above example this is trivial, since `Content` and `Site` are disjointed. The lookup gets more interesting with content beans which usually constitute a deeply nested class structure. Assume, you configured settings finders for `CMLinkable` and `CMTaxonomy`. If you invoke the settings service with a `CMTaxonomyImpl` bean, only the settings finder for `CMTaxonomy` is effective. There is no automatic fallback to the `CMLinkable` finder. If you need such a fallback, let your special settings finder extend the intended fallback finder and call its `setting` method explicitly.

The easiest way to provide a custom way of fetching settings for certain documents or even for objects that do not represent a `CoreMedia` document, is, to add a corresponding settings finder, that does the trick. Therefore, you should use `CoreMedia`'s Spring bean customizer, that you can use anywhere within your Spring application context as follows:

Example 4.23. Adding Custom Settings Finder

```
<beans>
  <customize:append id="mySettingsFinders" bean="settingsFinders">
    <map>
      <entry key="example.org.MyClass" value-ref="mySettingsFinder"/>
    </map>
  </customize:append>
</beans>
```

Via Spring you can configure one settings finder per class. This is a tradeoff between flexibility and simplicity which is sufficient for most use cases. However, on the

Java level the `SettingsServiceImpl` provides the method `addSettingsFinder(Class<?>, SettingsFinder)` which allows you to add multiple settings finders for a class.

Typed Settings Interfaces

The `SettingsService` is a powerful multi-purpose tool. However, genericity always comes at the price of abstraction. Assume, there is some business logic which is based on a domain specific interface `Address`:

```
public interface Address {
    String getName();
    String getCity();
}

public class Messages {
    public static String getHelloMessage(Address address) {
        return "Hello " +
            address.getName() +
            ", are you living in " +
            address.getCity() + "?";
    }
}
```

Example 4.24. Business Logic API

If your actual address data is provided by the `SettingsService`, it must be adapted to the address interface.

```
class SettingsBackedAddress implements Address {
    // [...] constructor and fields for service and provider bean
    public String getName() {
        return settingsService.setting("name", String.class, bean);
    }
    public String getCity() {
        return settingsService.setting("city", String.class, bean);
    }
}
```

Example 4.25. Settings Address Adapter

Cumbersome, isn't it? Especially, if the interfaces are larger or not yet final. Fortunately, you don't need to implement such interfaces manually, but `SettingsService.createProxy` does the job for you:

```
class MyCode {
    private SettingsService settingsService;

    void doSomething(BusinessBean beanWithSettings) {
        Address address = settingsService.createProxy(Address.class,
            beanWithSettings);
    }
}
```

Example 4.26. Address Proxy

```
String message = Messages.getHelloMessage(address);
}
}
```

Internally the default settings service intercepts the call to `getName()` and `getCity()`. The operation `getCity()` is translated to `settingsService.setting("city", String.class, bean)`. Note: The property name "city" will be derived from the operation `getCity()` in the interface. Be aware of this dependency when choosing names for your settings properties and for the operations of your business objects if you want to use the proxy mechanism.

Content types Requirements

The settings module makes use of the `localSettings` and `linkedSettings` properties of the `CMLinkable` content type. The `SettingsService` itself does not depend on particular content types, but some provided `SettingsFinder` implementations support these properties. If you need struct data which is not to be handled by the `SettingsService`, do not put it into `localSettings` and `linkedSettings`, but add new struct properties to the content type model.

`CMNavigation` documents inherit their settings along the hierarchy up to the root navigation. `CMTeaser` documents inherit the settings of their targets. If you rename these content types, this functionality gets lost.

4.3.2 Developing with Extensions

As described in [Section 4.1.5, "Project Extensions" \[114\]](#) the *CoreMedia Blueprint* workspace provides an easy way to enable or disable existing extensions in one place. This chapter shows you how to enable, disable or remove existing extensions in general and special.

- [Section "Adding, Disabling or Removing an Extension" \[147\]](#) describes how you manage extensions in general
- [Section "Removing the Elastic Social Extension" \[149\]](#) describes how to remove the *Elastic Social Extension*
- [Section "Removing the Adaptive Personalization Extension" \[149\]](#) describes how to remove the *Adaptive Personalization Extension*
- [Section "Removing the e-Commerce Blueprint" \[150\]](#) describes how to remove the *e-Commerce Blueprint*
- [Section "Removing the Brand Blueprint" \[150\]](#) describes how to remove the *Brand Blueprint*
- [Section "Removing the Advanced Asset Management Extensions" \[151\]](#) describes how to remove the *Advanced Asset Management Extension*

→ [Section “Extensions and Their Dependencies” \[151\]](#) lists all extensions and their mutual dependencies.

Adding, Disabling or Removing an Extension

This section explains the required steps to add, disable or remove an extension from the *CoreMedia Blueprint* workspace in general.

For most of the extensions, there is no single extension in the technical sense. See the Elastic Social extension, for instance. Instead, there is an integration, which consists of multiple extensions. The Elastic social integration, for example, consists of multiple Elastic Social extensions which augment other extensions, such as lc or p13n.

The reason for this is, that in order to use an extension generally, but disable some particular extensions (lc for example), it must be possible to disable only the lc related aspect of this extension. On the other hand, it must be possible to use the lc extension without the other extension. So, it is not possible, for example, to put all Elastic Social modules into a single extension nor include them directly in other extensions. Therefore, you have an integration which consists of several extensions.



Adding an Extension to the *CoreMedia Blueprint* workspace

To add an extension to *CoreMedia Blueprint* open the project's root POM and move to the `dependencyManagement` section. Import the Extension Descriptor (a POM import) as shown in [Example 4.27, “Activation of an Extension in the project's root POM” \[147\]](#).

```
<dependencyManagement>
  <dependencies>
    ...
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>my-blueprint-extension-bom</artifactId>
      <version>${project.version}</version>
      <scope>import</scope>
    </dependency>
    ...
  </dependencies>
</dependencyManagement>
```

Example 4.27. Activation of an Extension in the project's root POM

Update the extensions configuration with the following call in the root folder of the workspace :

```
mvn com.coremedia.maven:\
  coremedia-blueprint-maven-plugin:\
```

```
update-extensions
```

Disabling or Removing an Extension from the *Blueprint* workspace

The *CoreMedia Blueprint Maven Plugin* is a tool which supports developers to disable an extension in or remove an extension from the *Blueprint* workspace. Disabling an extension using this tool means that the provided feature will not be available and the source files are no longer part of the build process.

Removing an extension means, that the source code of the extension will be disabled and removed from the workspace as well.

Before you can remove an extension, you have to build the workspace once.



To disable an extension, open a console and go to the *Blueprint* workspace. Call the `disable-extensions` goal and identify the affected extension(s) by the Maven `artifactId` of the extension descriptor (a BOM POM) using the `core-media.project.extensions` option.

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
  disable-extensions \
  -Dcoremedia.project.extensions=my-extension.bom,\
  another-extension.bom
```

To remove an extension, use the `remove-extensions` goal of this Maven plugin instead of `disable-extensions`. The next sections describe how you remove the predefined CoreMedia extensions.

Preparing the Workspace for Further Development

As described in the documentation of the *CoreMedia Blueprint Maven Plugin* the POMs in the module `modules/extension-config` will be modified. The CoreMedia components in this workspace depend on these modules.

Therefore, when you want to start a web application like the CAE from the workspace using the Maven Tomcat 7 plugin, for example, the modified POMs must be available in the local Maven repository. To install them do the following:

```
$ cd $CM_BLUEPRINT_HOME/modules/extension-config
$ mvn clean install
```

During further development the CoreMedia components of the *Blueprint* workspace will consider the changed set of extensions, that is, added extensions will be enabled and removed extensions will no longer be available.

Removing the Elastic Social Extension

This section describes the required steps to remove the *CoreMedia Elastic Social* extension from *Blueprint*.

Removing the Elastic Social Integration is optional. Even if you have no licence or if you do not want to use it, you can leave the extension in the workspace.



1. Remove all affected extensions from the project. This includes all extensions from *Blueprint* whose name contains an *es* segment and the *shoutem* extension which depends on Elastic Social.

```
$ cd $BLUEPRINT_HOME
$ mvn
com.coremedia.maven:coremedia-blueprint-maven-plugin:remove-extensions \
  -Dcoremedia.project.extensions=es-bom, \
  lc-es-bom, ecommerce-ibm-es-bom, es-p13n-bom, es-alx-bom, shoutem-bom
```

Example 4.28. Remove CoreMedia Elastic Social Extension

2. Some documents of the demo content may have references to extension specific documents. After removing the extension, this may lead to errors in the CoreMedia components. The content has to be adapted not to use extension specific content any longer.

Either exclude any documents of types defined in `elastic-social-plugin-doctypes.xml` or manually add those content types to your *Content Server*.

3. Proceed with the instructions from [Section “Adding, Disabling or Removing an Extension” \[147\]](#) for further development.

Removing the Adaptive Personalization Extension

This section describes the required steps to remove the *CoreMedia Adaptive Personalization* extension from *CoreMedia Blueprint*.

1. Remove all affected extensions from the project. This includes *p13n*, *es-p13n*, *alx-p13n*, *lc-p13n* and *nuggad* which depend on *Adaptive Personalization*.

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
  remove-extensions \
  -Dcoremedia.project.extensions=p13n-bom, \
  alx-p13n-bom, es-p13n-bom, lc-p13n-bom, nuggad-bom
```

Example 4.29. Remove CoreMedia Adaptive Personalization Extension

2. The *Adaptive Personalization* integration brings along specific content types. If you disable the Adaptive Personalization Integration the *Content Server's* configuration will be affected as well. In this case you have to drop the databases of the *Content Servers* during the next deployment. Furthermore, you have to remove the documents of this content types from the demo content before importing or manually deploy the relevant content types to your content server.

Either exclude any documents of types defined in `personalization-doc types.xml` or manually add those content types to your *Content Server*.

Example 4.30. Example for Adaptive Personalization Content in Blueprint

3. Proceed with the instructions from [Section “Adding, Disabling or Removing an Extension” \[147\]](#) for further development.

Removing the e-Commerce Blueprint

This section describes the required steps to remove the *CoreMedia e-Commerce* Blueprint from the *CoreMedia Blueprint* workspace.

1. Remove the lc extension and its dependent extensions from the project.

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
  remove-extensions \
  -Dcoremedia.project.extensions=\
  lc-bom,lc-asset-bom,lc-es-bom,lc-pl3n-bom,\
  ecommerce-ibm-bom,ecommerce-ibm-es-bom
```

Example 4.31. Remove CoreMedia Livecontext Extension

2. Proceed with the instructions from [Section “Adding, Disabling or Removing an Extension” \[147\]](#) for further development.

Removing the Brand Blueprint

This section describes the required steps to remove the *CoreMedia Brand* Blueprint from the *CoreMedia Blueprint* workspace.

1. Remove the corporate extension and its dependent extensions from the project.

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
  remove-extensions \
  -Dcoremedia.project.extensions=\
  corporate-bom
```

Example 4.32. Remove CoreMedia Corporate Extension

2. Proceed with the instructions from [Section “Adding, Disabling or Removing an Extension” \[147\]](#) for further development.

Removing the Advanced Asset Management Extensions

This section describes the required steps to remove *Advanced Product Asset Management* from *CoreMedia Blueprint*. *Advanced Asset Management* consists of two extensions.

1. Remove the lc-asset and the am-bom extensions from the project.

```
$ cd $BLUEPRINT_HOME
$ mvn com.coremedia.maven:coremedia-blueprint-maven-plugin:\
  remove-extensions \
  -Dcoremedia.project.extensions=lc-asset-bom, am-bom
```

Example 4.33. Remove CoreMedia Product Asset Management Extension

2. If you use the CoreMedia example content you also have to remove the links to Asset content in the CMPicture files. You can use a tool like sed.
3. If you use the the new deployment scenario below deployment/chef, remove the dependency to am-adobe-drive-server-webapp from the pom.xml file in deployment/chef.
4. Proceed with the instructions from [Section “Adding, Disabling or Removing an Extension” \[147\]](#) for further development.

Extensions and Their Dependencies

This section sums up the existing extensions in *CoreMedia Blueprint* and shows their mutual dependencies. This information is required when removing extensions completely. The table describes the extension descriptors used to enable or disable the extension in the *CoreMedia Blueprint* workspace and lists extensions depending on the given one. These extensions have to be disabled as well when disabling the given extension.

You will also find a list of the extensions in the Blueprint workspace in the modules/extensions folder in the README.md file.

Table 4.5. Blueprint Extension Descriptors and Dependencies

alx-bom	
Description	General Analytics Integration
Required by Extension	alx-google, alx-webtrends, es-alx-bom, alx-p13n
alx-google, alx-webtrends	
Description	Specific Analytics Integration for Google Analytics and Webtrends. These extensions can be enabled or disabled independently.
Required by Extension	

alx-p13n-bom	
Description	Personalization plugin for Analytics, exposes personalization info (e.g. segment) for tracking
Required by Extension	alx-google, alx-webtrends, es-alx-bom
am-bom	
Description	<i>CoreMedia Asset Management</i> allows you to store digital assets (for example high-resolution pictures of products) in the content repository. An integration with Adobe Drive is available.
Required by Extension	
catalog-bom	
Description	Internal catalog.
Required by Extension	corporate-bom
corporate-bom	
Description	Extension with the features for the Brand Blueprint.
Required by Extension	
create-from-template-bom	
Description	Create a Page in Studio with predefined content.
Required by Extension	
custom-topic-pages-bom	
Description	Create custom topic pages in Studio.
Required by Extension	
ecommerce-ibm-bom	
Description	IBM WCS specific implementations and features for Livecontext
Required by Extension	ecommerce-ibm-es-bom
ecommerce-ibm-es-bom	
Description	IBM WCS specific Elastic Social implementations and features for LiveContext
Required by Extension	

es-bom	
Description	CoreMedia Elastic Social Integration
Required by Extension	es-p13n-bom, es-alx-bom, shoutem-bom, lc-es-bom
es-alx-bom	
Description	Extension to retrieve and cache computed data from Analytics. The data is persisted using CoreMedia Elastic Social.
Required by Extension	
es-controlroom-bom	
Description	Extension that enables the collaborative features and supports MongoDB as the database for collaborative components.
Required by Extension	
es-demodata-bom	
Description	Elastic Social Demo Data Generator. The Elastic Social Demo Data Generator is a tool to simulate actions of a website online community.
Required by Extension	
es-p13n-bom	
Description	Extension to retrieve data from CoreMedia Elastic Social for use in the Adaptive Personalization Extension.
Required by Extension	
external-library-bom	
Description	Adds the external library functionality to Studio. The external library is a Studio utility that supports to view external content, for example from an RSS feed, in Studio and create CMS content from the external content.
Required by Extension	
external-preview-bom	
Description	Adds the external preview functionality to Studio and CAE. The external preview is a Studio utility that supports to use one or more additional displays for Studio's preview.
Required by Extension	
memory-controlroom-bom	

Description	CoreMedia in-memory Control Room extension
Required by Extension	
lc-asset-bom	
Description	Feature allows you to manage images and image variants (or crops) for categories, products and products variants (products for short) in the CoreMedia system. These extensions depends on <i>CoreMedia LiveContext</i> .
Required by Extension	
lc-bom	
Description	Generic Livecontext Extension
Required by Extension	ecommerce-ibm, lc-es, lc-p13n, lc-asset
lc-es-bom	
Description	Elastic Social features for Livecontext
Required by Extension	ecommerce-ibm-es-bom
lc-p13n-bom	
Description	Personalization features for Livecontext
Required by Extension	
nuggad-bom	
Description	Nuggad Integration
Required by Extension	
optimizely-bom	
Description	Optimizely Integration
Required by Extension	
osm-bom	
Description	OpenStreetMap Integration
Required by Extension	
p13n-bom	
Description	CoreMedia Adaptive Personalization Integration

Required by Extension	es-p13n-bom, nuggad-bom, alx-p13n-bom, lc-p13n-bom
shoutem-bom	
Description	ShoutEm Integration
Required by Extension	
upload-bom	
Description	Studio Bulk Upload Integration
Required by Extension	

4.3.3 Extending Content Types

Developing a new software almost always starts by analyzing the domain model. This is not different for *CoreMedia CMS*. Here the domain model is the source for modeling the content type model. The content type model is the backbone of *CoreMedia CMS* as it describes what content means to you. Read section [Developing a Content Type Model](#) of the [Content Server Manual] for details on the content types.

Basically, there are two places within the *Blueprint* workspace you may use if you define your own content type model or extend the *Blueprint's* one. You will learn both of them by defining a new content type `CMHelloWorld` as a child of `CMTeaser` within a new file `mydoctypes.xml` as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<DocumentTypeModel
  xmlns="http://www.coremedia.com/2008/documenttypes"
  Name="my-doctypes">

  <ImportGrammar Name="coremedia-richtext-1.0"/>
  <ImportDocType Name="CMTeaser"/>

  <DocType Name="CMHelloWorld" Parent="CMTeaser">
    <StringProperty Name="message" Length="32"/>
  </DocType>
</DocumentTypeModel>
```

Defining content types in `contentserver-blueprint-component`

The first and a little easier way of defining `CMHelloWorld` is to put the new file `mydoctypes.xml` shown above into the directory `modules/server/contentserver-blueprint-component/src/main/resources/framework/doc types/my/`. It is good style to create a subfolder under `doctypes` for your customization, here named "my".

After doing so, you can test your new content type. To do so, you have to build the `contentserver-blueprint-component` module and the `content-management-server-webapp` module as follows. Remember to stop the server if you have not already.

```
$ cd modules/server/contentserver-blueprint-component
$ mvn clean install
$ cd modules/server/content-management-server-webapp
$ mvn clean install
```

Now, start the *Content Management Server* web application and take a look into its log file. You should see the following message, telling you that the *Content Server* created a new database table for the new content type.

```
[INFO] SQLStore - DocumentTypeRegistry: creating table:
CREATE TABLE CMHelloWorld( id INT NOT NULL, version INT NOT NULL,
isApproved TINYINT, isPublished TINYINT, editorId INT,
approverId INT, publisherId INT, editionDate DATETIME,
approvalDate DATETIME, publicationDate DATETIME,
"locale" VARCHAR(32), "masterVersion" INT, "keywords" VARCHAR(1024),
"validFrom" DATETIME, "validFrom tz" VARCHAR(30), "validTo" DATETIME,
"validTo tz" VARCHAR(30), "segment" VARCHAR(64), "title"
VARCHAR(512),
"teaserTitle" VARCHAR(512), "notSearchable" INT, "message"
VARCHAR(32),
PRIMARY KEY (id, version), FOREIGN KEY (id) REFERENCES
Resources(id))
```

Using a Separate Module in the Context of an Extension

The second possibility is the more flexible way. You build your own module in the context of an extension. The following steps assume that an extension module `my-extension` already exists and requires a new content type. Proceed as follows:

1. Create a new subfolder `my-extension-server` in the `modules/extensions/my-extension` directory.
2. Create a `pom.xml` file and add the following contents.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.coremedia.blueprint</groupId>
    <artifactId>my-extension</artifactId>
    <version>${project.version}</version>
    <relativePath>../pom.xml</relativePath>
  </parent>

  <artifactId>my-extension-server</artifactId>

  <properties>
```

```

<coremedia.project.extension.for>
  server
</coremedia.project.extension.for>
</properties>
</project>

```

3. Adjust the `groupId` and `artifactId` of the parent declaration according to your project settings.
4. Add this module's Maven coordinates to the Extension Descriptor of the extension.
5. Create the subfolder `src/main/resources/framework/doctypes/myextension`.
6. Copy the content type definition file from above into the folder created in the last step.
7. Refer to [Section 4.3.2, “Developing with Extensions” \[146\]](#) to enable the extension.

4.3.4 Developing with Studio

New *Studio* modules can be added to the project using the Blueprint extensions mechanism or by adding them as child modules to the `studio-plugins` module. This section describes how to add a new Studio module to the list of `studio-plugins`. Adding a new Studio module as an extension works the same way but requires additional Maven configurations.

Maven Configuration

First, add the new plugin to the modules section of the `studio-plugins pom.xml` file:

```

<modules>
  <module>taxonomy-studio-plugin</module>
  <module>pagegrid-studio-plugin</module>
  <module>contentchooser-studio-plugin</module>
  <module>dynamic-content-query-studio-plugin</module>
  <module>struct-editor-studio-plugin</module>

  <!-- add module -->
  <module>sample-studio-plugin</module>
</modules>

```

Next, create the child folder of the new module and its `pom.xml` file.

Example 4.34. POM file of a new Studio module

```

<project>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.coremedia.blueprint</groupId>
    <artifactId>studio-plugins</artifactId>
    <version>8-SNAPSHOT</version>
    <relativePath>../pom.xml</relativePath>
  </parent>

  <artifactId>sample-studio-plugin</artifactId>
  <packaging>jangaroo</packaging>

  <description>Sample Studio Plugin for the CoreMedia
Studio</description>

  <dependencies>

    <dependency>
      <groupId>com.coremedia.ui.toolkit</groupId>
      <artifactId>ui-components</artifactId>
    </dependency>

    <dependency>
      <groupId>com.coremedia.ui.sdk</groupId>
      <artifactId>editor-components</artifactId>
    </dependency>

    <dependency>
      <groupId>net.jangaroo</groupId>
      <artifactId>ext-as</artifactId>
    </dependency>

  </dependencies>

  <build>
    <sourceDirectory>src/main/joo</sourceDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
        <includes>

<include>META-INF/resources/joo/${project.artifactId}.module.js</include>

        </includes>
      </resource>
      <resource>
        <directory>src/main/resources</directory>
        <filtering>false</filtering>
        <excludes>

<exclude>META-INF/resources/joo/${project.artifactId}.module.js</exclude>

        </excludes>
      </resource>
      <resource>
        <directory>target/generated-resources</directory>
      </resource>
    </resources>

    <plugins>
      <plugin>

```

```

<groupId>net.jangaroo</groupId>
<artifactId>jangaroo-maven-plugin</artifactId>
<extensions>>false</extensions>
<executions>
  <execution>
    <goals>
      <goal>properties</goal>
    </goals>
    <configuration>
      <resourceDirectory>src/main/joo</resourceDirectory>
    </configuration>
  </execution>
</executions>
</plugin>

<plugin>
  <groupId>net.jangaroo</groupId>
  <artifactId>exml-maven-plugin</artifactId>
  <version>${jangaroo.version}</version>
  <extensions>>true</extensions>
  <configuration>

<configClassPackage>com.coremedia.blueprint.studio.config.contentchooser</configClassPackage>
  </configuration>
</plugin>
</plugins>
</build>
</project>

```

The module contains the packaging `jangaroo` which is necessary so that during the Maven build time, the module is recognized as a Studio module.

Your IDE should display the new plugin as a successfully linked Maven module. Now the module must be added as a dependency to the *Studio* web application. Open the `pom.xml` of the `studio-webapp` module and add following dependency:

```

<dependency>
  <groupId>${project.groupId}</groupId>
  <artifactId>sample-studio-plugin</artifactId>
  <version>${project.version}</version>
  <scope>runtime</scope>
</dependency>

```

Your module should have the following structure by now:

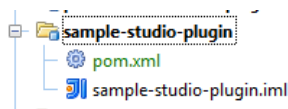


Figure 4.4. The new sample studio plugin

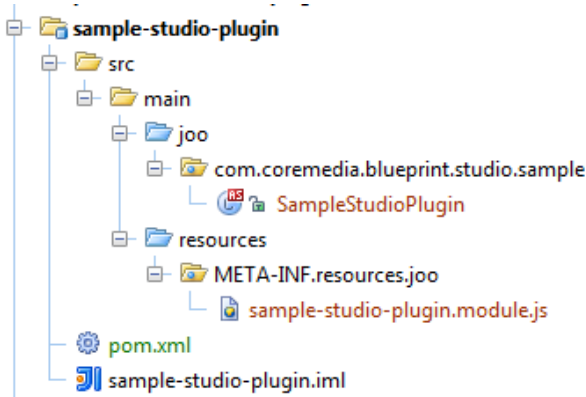
Source Files and Folders

A Studio module contains at least two files: the plugin descriptor file located in the resource folder (`sample-studio-plugin.module.js`) of the module and the initializing plugin class (`SampleStudioPlugin.as`).

Ensure that the name prefix of the plugin descriptor matches the name of the Maven module.



Figure 4.5. The sample studio plugin with plugin class and descriptor



Add the class to the corresponding package, created in the subfolder `src/main/joo`. You can use the IDEA context menu to apply the newly created `joo` folder as the source path. It shows the formatted package name of the class afterwards. Create an additional source path `resources` next to the `joo` folder and create the subfolders `META-INF/resources/joo`. Create the file `sample-studio-plugin.module.js` afterwards.

The plugin class only implements the `init` method of the `EditorPlugin` interface:

```

package com.coremedia.blueprint.studio.sample {
import com.coremedia.cms.editor.sdk.EditorPlugin;
import com.coremedia.cms.editor.sdk.IEditorContext;

public class SampleStudioPlugin implements EditorPlugin {
    public function init(editorContext:IEditorContext):void {
    }
}

```

The plugin descriptor class is read when the Studio's web page is invoked. It contains the class name of the plugin class and can declare additional CSS files or other resources required for the plugin.

```

joo.loadModule("${project.groupId}", "${project.artifactId}");
coremediaEditorPlugins.push({
name: "Sample Plug-in",
mainClass: "com.coremedia.blueprint.studio.sample.SampleStudioPlugin"
});

```


The object pushed onto the array `coremediaEditorPlugins` may use the attributes defined by the class `EditorPluginDescriptor`, especially `name` and `mainClass` as shown above. In addition, the name of a group may be specified using the attribute `requiredGroup`, restricting access to the plugin to members of that group.

When set up correctly, your project structure should compile successfully.

Additional steps would be adding resource bundles and plugin rules to your plugin. For more details about this and developing Studio plugins and property editors have a look at the [Studio Developer Manual].



Joo Unit Testing

The *jangaroo-maven-plugin* supplies unit testing of `ActionScript` classes inside the Studio modules. Please refer to [Jangaroo Tools Wiki, chapter "Unit Testing"](#). The tests can be triggered by following applicable conditions:

- By stating the Maven profile "joo-unit-tests" explicitly
- By stating the property `phantomjs.bin` with suitable value leading to the *phantomjs* bin path
- *phantomJS* is located in `/usr/local/bin/phantomjs` on Linux/Unix OS

The test results are gathered in the `/target/surefire-reports` directory of modules containing Joo unit tests. To debug the Joo unit tests, run

```
mvn jangaroo:jetty-run-tests
```

within the appropriate module and open the URL listed below with the browser.

4.3.5 Developing with the CAE

The CAE can be extended with new capabilities by using the *Blueprint* extension mechanism or by just creating a new module with the required resources. In both cases the extension will be activated by adding a Maven dependency on the new module. This section describes how to add a new *Blueprint* module which contains an additional view template and a new view repository using this template.

Maven Configuration

First you have to create a new module which contains the required resources. The location of the new module inside the workspace is not important to enable the new features provided by the module. But to keep cohesion in the aggregation modules of the *CoreMedia Blueprint* workspace the new module should be created

next to other CAE functions. In this example the new module `sample-cae-extension` will be created in the `modules/cae` module.

1. First, add a new module entry named `sample-cae-extension` to the `modules` section of the `cae pom.xml` file:

```
<modules>
  <module>cae-base-lib</module>
  <module>cae-base-component</module>
  <module>cae-live-webapp</module>
  <module>cae-preview-webapp</module>
  <module>contentbeans</module>
  <module>cae-performance-test</module>
  <module>cae-test</module>

  <!-- add module -->
  <module>sample-cae-extension</module>
</modules>
```

2. After that create a new subdirectory `sample-cae-extension` and add the `pom.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
  <project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
      <groupId>com.coremedia.blueprint</groupId>
      <artifactId>cae</artifactId>
      <version>BLUEPRINT_VERSION</version>
      <relativePath>../pom.xml</relativePath>
    </parent>

    <artifactId>sample-cae-extension</artifactId>
    <packaging>jar</packaging>

    <dependencies>
      <dependency>
        <groupId>com.coremedia.cms</groupId>
        <artifactId>coremedia-spring</artifactId>
        <scope>runtime</scope>
      </dependency>
    </dependencies>

    </project>
```

Now the basic structure for the extension exists.

Enabling the Extension

To enable the extension the target component has to depend on the created extension module.

To enable the new capabilities in all CAEs add the following dependency to the `pom.xml` of the `cae-base-component` module:

```
<dependency>
  <groupId>${project.groupId}</groupId>
  <artifactId>sample-cae-extension</artifactId>
  <version>${project.version}</version>
</dependency>
```

Creating Source Files and Folders

The sample extension for the CAE provides a new view template for the content type `CMArticle` to display external content and a new view repository configuration which includes this view template.

1. Create the new view template `CMArticle.jsp` in the module `sample-cae-extension` in the directory `src/main/resources/META-INF/resources/WEB-INF/templates/external-content-view-repository/com.coremedia.blueprint.common.contentbeans`.
2. To include the new view repository add a new file `component-sample-cae-extension.xml` to the directory `src/main/resources/META-INF/coremedia`. Add the following contents to the file:

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:customize="http://www.coremedia.com/2007/coremedia-spring-beans-customization"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.coremedia.com/2007/coremedia-spring-beans-customization
      http://www.coremedia.com/2007/coremedia-spring-beans-customization.xsd">

    <customize:prepend id="addSampleViewRepository"
      bean="viewRepositories">
      <description>
        Add repository name, relative to /WEB-INF/templates/
      </description>
      <list>
        <value>sample-cae-extension</value>
      </list>
    </customize:prepend>
  </beans>
```

Running performance tests

In order to make performance testing as easy as possible, the workspace includes a module with a preconfigured JMeter test, `cae-performance-test`.

Module description

An example module for a ready-to-run performance test for CAE web applications.

It uses JMeter as well as a Maven plugin for analyzing and providing results. To keep the example as simple as possible, it is designed to run against the *Preview CAE*. Typically, *Live CAEs* running against a *Replication Live Server* are performance tested on appropriate (live) hardware from a remote machine. Also, plugin configuration and executions are configured in a profile `performance` so that no performance test is executed when building the workspace with `mvn package` or `mvn install`.

The module uses the [jmeter-maven-plugin](#) to run the JMeter test and the [jmeter-analysis-maven-plugin](#) to analyze the results. It can analyze JMeter result files and produces a result HTML, easy to read images with result data and summary CSV files.

Prerequisites

If the performance test is used as is, the following prerequisites must be fulfilled:

- ➔ The *Content Management Server* must be started
- ➔ The *Preview CAE* must be started

Execution

To execute the performance test, run

```
$ mvn verify -Pperformance
```

Configuration

If you want to know more about configuration possibilities,

- ➔ Have a look at the `pom.xml` file where every used configuration element is fully documented
- ➔ If you want to run your tests against another host and port just override the properties `webapp.host` and `webapp.port`, for example `mvn verify -Pperformance -Dwebapp.host=myHost -Dwebapp.port=4711`
- ➔ To use individual URLs for the performance test, you can provide an arbitrary file with URL lists and configure the test plan to use it with the property

`webapp.uris`, for example `mvn verify -Pperformance -Dwebapp.uris=path-to-uri-file`. The file contains a list of URI paths; that is without protocol, host and port information.

Test URLs

To automatically generate test URLs for the CAE, the `SiteMapController` can be used. See section [Sitemap](#) in the [Blueprint Concepts Guide].

To manually generate a list of test URLs, spider the site you want to test and save the URIs.

4.3.6 Customizing the CAE Feeder

Before customizing the *CAE Feeder*, you should be familiar with the content of [Section 4.3.5, “Developing with the CAE” \[161\]](#) about the CAE modules. Details about how the *CAE Feeder* works and how it may be customized are presented in the [Search Manual].

4.3.7 Adding Common Infrastructure Components

CoreMedia applications share common infrastructure components. CoreMedia provides common application infrastructure components for logging and JMX management. An application might use this infrastructure by simply adding the particular component artifact to the application by defining a Maven dependency. There is also the Base component, that aggregates basic infrastructure to be used by all components.

The Logging Component

This component provides a common logging infrastructure based on the [logback](#) framework. The component adds some new features:

- ➔ Automatic registration of a configuration MBean as `com.core-media:Type=Logging,application=<applicationname>`
- ➔ A fallback logging configuration that is loaded when the application doesn't provide a more specific one. You will find this configuration in the file `/META-INF/resources/WEB-INF/logback-default.xml` in the `logging-component.jar` file.
- ➔ Preconfigured logging appenders that can be simply imported from log configuration files. Two profiles (“development” and “production”) have been defined that hold slightly different appender configurations. The “development” profile logs more information for development issues while the “production” profile should be used in a live system.

Adding the Logging Component

You can simply add the CoreMedia logging component to your CoreMedia web application as described below. For your custom services and code, you should use [slf4j](#) as a logging interface. When using the logging component, slf4j will be automatically bound to the logback logger implementation.

Adding dependency

Adding the component simply consists of adding the following dependency to your web application's project:

```
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>logging-component</artifactId>
  <scope>runtime</scope>
</dependency>
```

Example 4.35. Maven Dependency for Logging

Adding log configuration

If you want to use your own log configuration, add a file `/WEB-INF/logback.xml` to your application that looks like the following:

```
<configuration>
  <!--
    includes a common configuration that dispatches to
    "coremedia-development-profile.xml"
    or
    "coremedia-production-profile.xml" depending
    on property "coremedia.logging.profile".

    You can define a custom log pattern ("log.pattern") or a custom
    log file name ("log.file") like this:
    <property name="log.pattern" value="%d %-7([%level]) %logger -
    [%X{tenant}] %message \\\(%thread\\)\%n" />

    -->
    <include resource="logging-common.xml"/>

  <!-- adds project specific logger and references a common appender
  -->
  <logger name="com.coremedia" additivity="false" level="info">
    <appender-ref ref="file"/>
  </logger>

  <root level="warn">
    <appender-ref ref="file"/>
  </root>
</configuration>
```

Example 4.36. Logback Configuration

Changing log directory

By default, logs are written in the `./logs` directory. If you want to change this directory you have to either pass a system property `coremedia.logging.directory` or a JNDI property `java:comp/env/coremedia/logging/directory`

to the application. For example, when you use Tomcat you can add the following code to `context.xml`:

```
<Context ...>
...
  <Environment name="coremedia/logging/directory"
    value="/my-log-dir" type="java.lang.String" override="true"/>
...
</Context>
```

Example 4.37. Change Log Directory in Tomcat

See the [Tomcat documentation](#) for more details.

Switching the log profile

If you want to switch the logging profile from the default "production" to "development", you have to either set the system property `coremedia.logging.profile` or the JNDI property `java:comp/env/coremedia/logging/profile` to the "development" value.

Changing log configuration at runtime

If you want to change the logback configuration during runtime, there are two options: You can either use JMX or logback's automatic reloading mechanism. To let logback reload the configuration, you have to add the `scan` attribute to the configuration element.

```
<configuration scan="true" scanPeriod="30 seconds">
...
</configuration>
```

Example 4.38. Automatically reload configuration file every 30 seconds



Consider that the `scan` feature only works for configuration loaded from the file system. It does not work for classpath or web application resources. To this end you have to override the property `coremedia.logging.configuration` and provide a file URL pointing to the logback configuration (such as `file:/path/to/logback.xml`). The property can be set via JNDI or as system property.

The JMX Component

This component provides a common JMX infrastructure with the following features:

- ➔ All beans which are added to the map `mbeans` will be exported as MBeans to an MBean server
- ➔ An MBean remote connector server (and a `RMIRRegistry` if necessary) is started if a JMX service URL is specified

- ➔ MBeans are exported automatically using a completed object name

The component is preconfigured in *CoreMedia Blueprint* but does not use the own remote connector server. Instead, the container's remote connector server is used which is the recommended way for *CoreMedia DXP 8*.

Adding the JMX Component

If you want to add the JMX component to your own web application project, proceed as follows:

Adding JMX

1. Add the following dependency to your web application project:

```
<dependency>
  <groupId>com.coremedia.cms</groupId>
  <artifactId>management-component</artifactId>
  <scope>runtime</scope>
</dependency>
```

Example 4.39. Dependency for JMX

2. Add a property `management.server.remote.url` to `/WEB-INF/application.properties` to your application. The value of the property is the URL of the component's server. For example:

➔ `service:jmx:rmi://localhost/jndi/rmi://localhost:1098/myapplication`

This will start the adequate remote connector server so that the application's MBeans are available under the specified URL. If you want to use Tomcat's server, read the paragraph ["Using Tomcat's remote connector server"](#).

3. Every component has to register its MBeans by itself in order to make its MBeans available to the management component. Therefore, add a configuration like the following to the components descriptor in `/META-INF/coremedia/component-<component-name>.xml`.

```
<import
  resource="classpath:/com/coremedia/jmx/mbean-services.xml"/>

<bean id="myComponentMbeanRegistrar"
      class="com.coremedia.jmx.MBeanRegistrar">
  <property name="registry" ref="mbeanRegistry"/>
  <property name="mbeans">
    <map>
      <entry key="type=MyService" value-ref="myBean"/>
    </map>
  </property>
</bean>
```

Example 4.40. Register the MBeans

The MBean's object name will be automatically completed, a configured name "type=MyService" will, for instance, be automatically transformed to `com.coremedia:type=MyService,application=<applicationname>`

Using Tomcat's remote connector server

Instead of starting a custom remote connector server you might also use Tomcat's remote connector server infrastructure. In this case, leave the property `management.server.remote.url` empty and pass the following properties to Tomcat's `catalina.bat/catalina.sh` file:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=8008
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

Example 4.41. Use Tomcat remote connector server

If you require authentication add and change the properties as follows and provide the appropriate access and password file:

```
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.password.file= \
  ../conf/jmxremote.password
-Dcom.sun.management.jmxremote.access.file= \
  ../conf/jmxremote.access
```

Example 4.42. Use Tomcat remote connector server with authentication

See also [Enabling_JMX_Remote in Tomcat documentation](#) and [JmxRemoteLifecycleListener in Tomcat documentation](#) for how to enable JMX for Tomcat.

Now, you can reach Tomcat's remote connector via

```
service:jmx:rmi:///jndi/rmi://localhost:8008/jmxrmi
```

The Base Component

This component aggregates basic infrastructure to be used by all components. It contains a dependency on the Logging and JMX component and provides the mechanisms for bootstrapping all other components. It also implements configuration file and properties loading scheme described in [Section 4.1.3, "Application Architecture"](#) [107].

Adding the Base Component

To use the base component, add the following dependency to your component or web application module `pom.xml`:

Example 4.43. Adding the Base Component

```
<dependency>
<groupId>com.coremedia.cms</groupId>
<artifactId>base-component</artifactId>
<scope>runtime</scope>
</dependency>
```

Most applications (like the *Content Application Engine*) need an external service like the *Content Management Server* to startup. In order to make such an application start up independently, the spring application context is loaded asynchronously. Incoming requests must wait until the Spring root context is initialized.

This behavior can be disabled at runtime by configuring

```
com.coremedia.springframework.web.context.disableAsynchronousLoading=true
```

by either setting a servlet context init param, a servlet config init param, or any other property source known to the [Standard Servlet Environment](#).

4.3.8 Managing Properties in the Workspace

One wants to add properties to the workspace either in new components or existing code. This chapter describes, how to handle these properties from the code to the packaging and ensure the replacement of the property with default values as well as dynamic substitutions during the installation of RPMs.

Usage of Properties in Components

The correct usage and substitution of properties will be demonstrated by an example. Assume that the property `example.value` is introduced in the source code and your task is to assign the value `120` to it.

Properties for components, which are designed to be changeable, should be propagated via a property file in the web applications. In this case there is a property file in the *Preview CAE* called `modules/cae/cae-preview-webapp/src/main/webapp/WEB-INF/application.properties` where properties defined by the components themselves are overwritten. Thus, add the following line to `application.properties`:

```
example.value=120
```

This definition has to be added to every web application the plugin belongs to, but can have different values. The web application can be started with these settings via `mvn tomcat7:run`, for example.

Defining default values for deployment

To make the introduced property deployment independent, you have to configure the property in the `packages/` part of the workspace too. Under `services/`,

there are the deployment services like the `studio-tomcat` or `delivery-tomcat`.

To configure the new property add the following value for the preview web application in `studio-tomcat/src/main/override-properties/blueprint/WEB-INF/application.properties`.

```
example.value=@EXAMPLE_VALUE@
```

This is the base configuration for further deployment independent configurations. To set the default values for this property, you have to set the desired value in `packages/src/main/filters/preconfigure.properties` with the value:

```
EXAMPLE_VALUE=120
```

The token within the configuration of the web application will be replaced with these values in case of preconfigured builds of the packages.

Configuring this property for post-configured RPM deployment

To make this variable configurable on different deployment environments, set the new property in the file `packages/src/main/filters/postconfigure.properties` in the following way:

```
EXAMPLE_VALUE=@configure.EXAMPLE_VALUE@
```

The property `configure.EXAMPLE_VALUE` can be set on the target server to a server specific value, and will substitute this configuration in the RPM while installing it. Default values can also be provided by set the value in `packages/src/main/filters/default-deployment.properties`. If the property is not configured on the target server, this default value will be applied instead.

Defining default value

To get familiar with this mechanism, you should take a look at the predefined properties all over the workspace, components and modules and read the [Section 4.3.9, "Configure Filtering in the Workspace" \[171\]](#) for a deeper introduction into the configuration process. If you use the post-configuration approach and use the RPM files for installation, the information about missing properties will be logged in the system log at `/var/log/messages`.



4.3.9 Configure Filtering in the Workspace

Speaking about deployment, one has to choose between configuration at build time, that is preconfiguration, and configuration at installation time, that is post-configuration. Regardless of which approach you choose, you have to add filtering to your build repertoire. In a preconfiguration build, a filter token, called the source token, is being replaced with an explicit value, whereas in a post-configuration build, the source token is being replaced with another filter token, called the target

Postconfiguration vs. preconfiguration

token. The target token will be replaced not during build but during installation time on the target machine.

Replacing a token with a token may seem strange at first sight but it is necessary in order to provide a central configuration facility that can deal with both approaches, preconfiguration and post-configuration.

As a convention, the target token is derived from the source token by adding the common prefix "configure.". To distinguish the source tokens and target tokens from any other property that can be used in Maven for filtering and other purposes, they are all defined uppercase with underscores as a separation element.

Token naming scheme

Central configuration files

In Maven, a central configuration facility can be either the main `pom.xml` file or some property files to be loaded by a plugin. In the *CoreMedia DXP 8* workspace the latter approach is used and you can find the three property files below `packages/src/main/filters`:

- `default-deployment.properties`
- `postconfigure.properties`
- `preconfigure.properties`

The files `postconfigure.properties` and `preconfigure.properties` contain the properties for build-time filtering whereas the `default-deployment.properties` file contains the default values for filtering at installation.

By default, the *CoreMedia Blueprint* workspace uses the post-configure approach, which means, that it loads the properties found in the `postconfigure.properties` for build-time filtering. If you want to use the preconfigure approach, you simply have to enable the `preconfigure` Maven profile.

Preconfigure profile

Configuration templates

To allow reconfiguration without redeployment for post-configuration builds, all deployable artifacts from the packages hierarchy contain their files with target tokens twice. The copies of these files are packaged below an `INSTALL` directory as an overlay to the installation root. At reconfiguration time the overlay is copied on top of the application and in a second step all target tokens are being replaced with the values found in the applications configuration file.

In addition to the configuration templates, the *coremedia-application-maven-plugin* selects all those properties from the `default-deployment.properties` file that match the target tokens found in the configuration template and generates a default configuration file, containing all tokens necessary to post-configure this package. The default configuration file is generated in a modules output directory

below the `INSTALL` folder with the name of the application as the filename. You can generate a complete list of these files in the `boxes/target/configuration-templates` folder, if you execute one of the update scripts in the `boxes` folder after you build the packages hierarchy.

To support usability at deployment, the commented descriptions of the target tokens in the `default-deployment.properties` file are being merged into the default configuration files too. As a result, adding a new source token to a configuration file requires the following:

→ add the source token with an explicit value to the `preconfigure.properties`

or

→ add the source token to target token mapping to the `postconfigure.properties`

→ add the target token with a good default value to the `default-deployment.properties`

Build verification

To prevent you from deploying packages with undetected tokens, the *coremedia-application-maven-plugin* will fail the build, if target tokens are found not starting with the common prefix. The pattern for this check can be configured in the plugin configuration in the `packages/pom.xml` with the element `allowFilterTokens`.

Override properties

Filtering requires the presence of ant-style filter tokens in the configuration files and would therefore require the duplication of the whole configuration file even if only one value has to be replaced. With CoreMedia 7 some filtering enhancements have been added to the *coremedia-application-maven-plugin* to define only those properties that you want to reconfigure. In the workspace you can find these deltas as property files below the `override-properties` directory. If you want to override a file completely, you can still do this by adding files below the `src/main/app` directory. For a more detailed description how to use and configure this mechanism, you should read the plugins [documentation](https://documentation.coremedia.com/utilities/coremedia-application-maven-plugin/2.6/) at <https://documentation.coremedia.com/utilities/coremedia-application-maven-plugin/2.6/>.

5. IBM WebSphere Commerce Integration

This chapter describes how the CoreMedia system integrates with *IBM WebSphere Commerce Server*. You will learn how to add fragments from the CoreMedia system into a WebSphere generated site, how to access the IBM WebSphere catalog from the CoreMedia system and how to develop with the *e-Commerce API*. The configurations of your IBM RAD system are described in [Section 3.4, “Customizing IBM WebSphere Commerce” \[58\]](#)

In general *CoreMedia Digital Experience Platform 8* offers two integration scenarios with IBM WebSphere Commerce: Content-led (see [Section 5.2.1, “Content-led Integration Overview” \[195\]](#)) and commerce-led (see [Section 5.1, “Commerce-led Integration Scenario” \[176\]](#)).

Integration scenarios

- In the commerce-led scenario, pages are delivered by the WCS. The page navigation is determined by the catalog category structure and cannot be changed in the CMS. You can augment the categories and product detail pages with content from the CMS. Content and settings are also inherited along the catalog category structure.
- In the content-led scenario, pages are delivered by both systems, transparent for the user. You can manipulate the navigation through the catalog pages and add complete new navigation paths. You can augment product detail pages with content from the CMS. Categories are rendered from the CAE. However, content and settings are inherited along the catalog category structure. [Figure 5.1, “The CoreMedia Perfect Chef site with dynamic price information from the IBM WebSphere Commerce shop” \[175\]](#) shows a page delivered by the CMS but enhanced with price information from the WCS system.

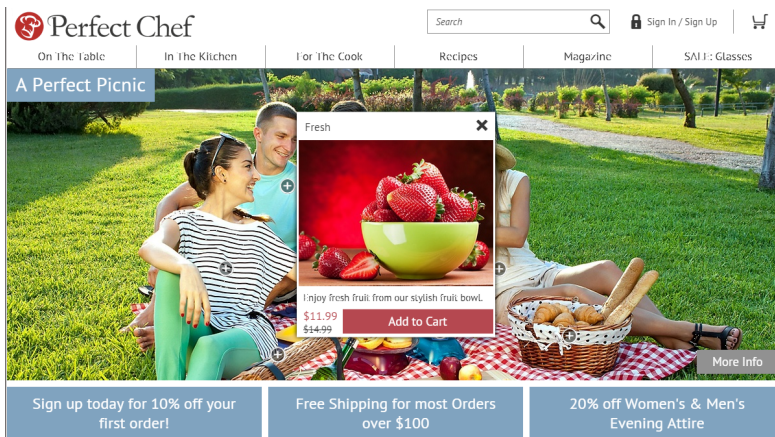


Figure 5.1. The CoreMedia Perfect Chef site with dynamic price information from the IBM WebSphere Commerce shop

- Section 5.1, “Commerce-led Integration Scenario” [176] describes the commerce-led scenario and shows how you extend WCS pages with CMS fragments.
- Section 5.2, “Content-led Integration” [195] describes the content-led scenario and some content-led specific configurations.
- Section 5.3, “Communication” [204] describes the communication between the IBM WCS and the CoreMedia CMS system.
- Section 5.4, “Connecting with an IBM WCS Shop” [207] describes how you connect a CoreMedia web application with an IBM WebSphere Commerce store.
- Section 5.5, “Link Building for Fragments” [213] describes deep links from fragments of the CMS system to pages of the WCS system.
- Section 5.6, “Enabling Preview of Commerce Category Pages in Studio” [215] describes how you activate the preview of WCS pages in *Studio*.
- Section 5.7, “Enabling Contract Based Preview” [216] describes how you enable the preview of WCS content based on contracts.
- Section 5.8, “The e-Commerce API” [219] describes main classes of the CoreMedia e-Commerce API, which you can use to access the WCS system.
- Section 5.9, “Commerce Cache Configuration” [221] describes the CoreMedia cache for e-Commerce entities from the WCS system.
- Section 5.10, “Studio Integration of the IBM WebSphere Commerce Content” [223] shows the e-Commerce features integrated into *CoreMedia Studio*.

5.1 Commerce-led Integration Scenario

In the commerce-led integration scenario, the WCS delivers content to the customer. The WCS pages are augmented with fragment content from the CoreMedia system.

This chapter describes how you include the content from the CMS into the WCS. Have also a look into [Section 5.10.5, “Augmenting WCS Content” \[231\]](#) and Chapter 6, *Working with Product Catalogs in CoreMedia Studio User Manual* for more details about the *Studio* usage for e-Commerce.

- [Section 5.1.1, “Commerce-led Integration Overview” \[176\]](#) gives an overview over the request flow in the commerce-led integration scenario.
- [Section 5.1.2, “Solutions for Same-Origin Policy Problem” \[177\]](#) describes how the same-origin policy problem has been solved for the CoreMedia solution.
- [Section 5.1.3, “Extending the Shop Context in Commerce-led Integration Scenario” \[181\]](#) describes how you extend the shop context that is delivered to the CMS.
- [Section 5.1.4, “Extending with Fragments” \[183\]](#) describes how you can add fragments to the WCS via the CoreMedia widgets and the `lc:include` tag and how you augment the WCS pages in *Studio*.

5.1.1 Commerce-led Integration Overview

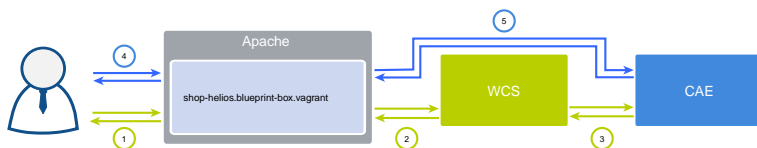


Figure 5.2. Commerce-led integration scenario

Figure 5.2, “Commerce-led integration scenario” [176] shows the commerce-led integration scenario where the CoreMedia CAE operates behind the commerce server. Moreover, you can see two kinds of requests. While the green arrows represent HTTP page requests to the commerce server, that include fragments delivered by the CAE, the blue arrows are resource or Ajax requests directly redirected by the one virtual host in front of both servers to the CAE.

A typical flow of requests through a commerce-led system is as follows:

1. A user requests a product detail page that is received by the virtual host.
2. The virtual host identifies the request as a commerce request and forwards it to the commerce server.

- 3. Part of the requested Product Detail Page (PDP) is a CAE fragment as described in the corresponding Commerce Composer page layout. Hence, the WCS requests the fragment from the CAE.
- 4. The resulting HTML page flows back to the user's browsers. Because the page contains dynamic CAE fragments which have to be fetched via Ajax, the browser triggers the corresponding request, against the virtual host.
- 5. As this is a CAE request, the virtual host forwards it directly to the CAE.

From the point of view of the user all requests are sent to exactly one system, represented by the one virtual host that forwards the requests accordingly.

5.1.2 Solutions for Same-Origin Policy Problem

When the commerce system has to deliver the end user's web pages, *CoreMedia Digital Experience Platform 8* offers a way to enrich those web pages with content from the CoreMedia CMS; the fragment connector. [Figure 5.3, “The Perfect Chef header as a fragment for the Aurora shop” \[177\]](#) shows the IBM Aurora demo shop page with the integrated CoreMedia PerfectChef header.

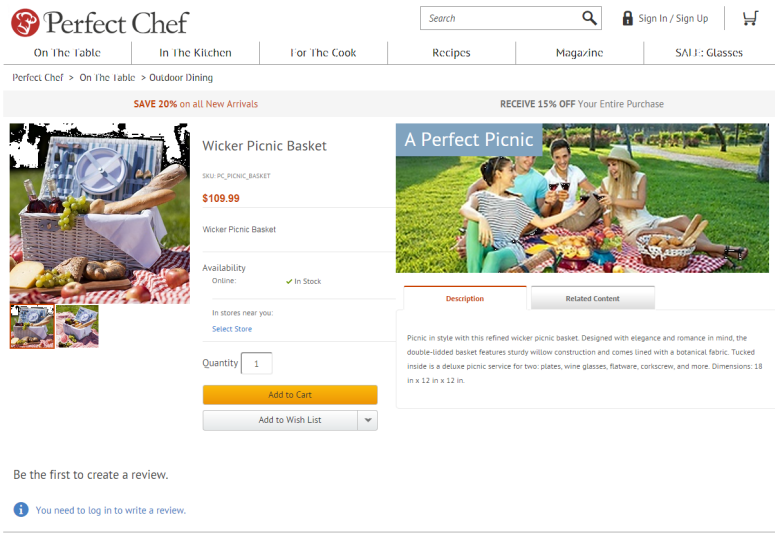


Figure 5.3. The Perfect Chef header as a fragment for the Aurora shop

Integrating content from the CoreMedia system into the IBM WebSphere Commerce pages presents a challenge due to the same-origin policy:

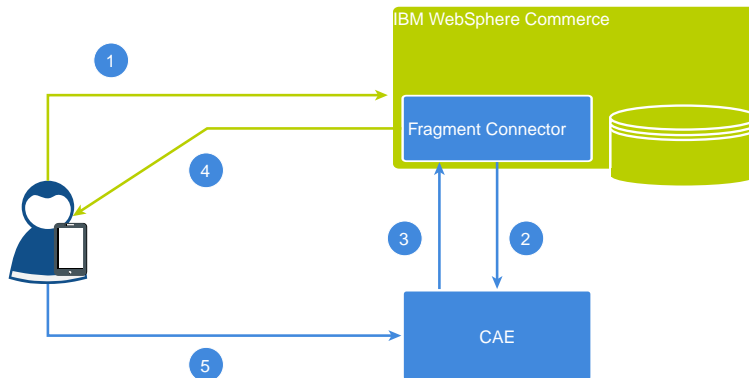


Figure 5.4. Cross Domain Scripting with Fragments

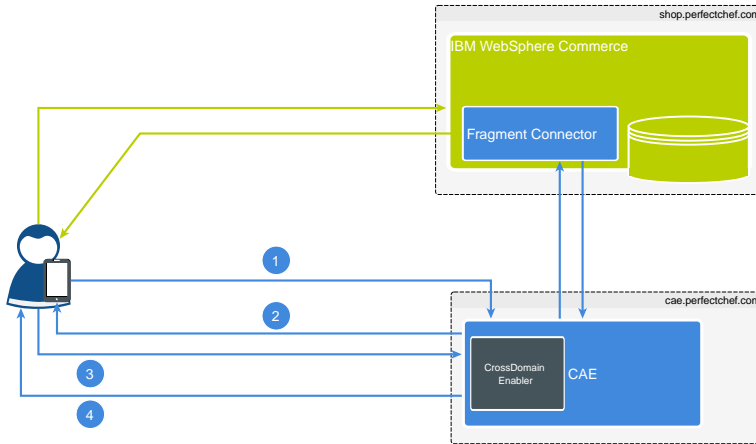
The image above shows a typical situation when a user requests an IBM commerce page that includes CoreMedia fragments.

1. The page request from the end user is sent to the IBM WebSphere Commerce server.
2. While rendering the page, the commerce server requests a fragment from the CAE.
3. The returned fragment contains itself parts that must be delivered dynamically. Take the login button. It is user specific, hence it must not be cached. The CoreMedia LiveContext Blueprint may include such parts via Ajax requests or as ESI tags, depending on the capabilities of the component which sent the request.
4. The commerce server returns the complete page, including the fragment that was rendered by the CAE.
5. Because it is assumed that the *CoreMedia LiveContext* fragment contains a dynamic part, which must not be cached, the browser tries to trigger an Ajax request to the CAE. But this breaks the same-origin policy and will not succeed.

Solution 1: Access-Control-Allow-Origin

The first solution is built into the CoreMedia Blueprint workspace, so you may use it out of the box. The idea is to customize the same origin policy by setting the `Access-Control-Allow-Origin` HTTP header accordingly.

Figure 5.5. The *CrossDomainEnabler*



CoreMedia Blueprint contains a servlet filter that does this job: The *CrossDomainEnabler*. The picture shows the relevant parts of the request flow:

1. For every Ajax request that is called from within a fragment, that was delivered by the commerce system, the JavaScript will trigger an `HTTP OPTIONS` request, asking for the allowance to set the `X-Requested-With` header.
2. The *CrossDomainEnabler* receives that request and allows that header to be sent for the given Ajax request.
3. The JavaScript client triggers the Ajax request and sets the `X-Requested-With` header to `XMLHttpRequest`.
4. The *CrossDomainEnabler* intercepts this request and writes its cross domain whitelist to the `Access-Control-Allow-Origin` header of the response before it forwards the request to the corresponding handler.
5. The browser receives the response and accepts that it.

The *CrossDomainEnabler* does not allow every cross domain access. Instead, it must be configured within the Spring application context with domains that are meant to be safe. To customize the *CrossDomainEnabler*, define the property `livecontext.crossdomain.whitelist` in the application context with a comma separated list of domains, as in the example below.

```
livecontext.crossdomain.whitelist=http://my.shop.domain1,http://my.shop.domain2
```

There are two additional properties that you may want to change.

1. The `ajaxIndicatorHeaderName` is the name of a header that the cross domain enabler uses to identify Ajax requests. Its default value is `X-Requested-With` which is used by jQuery to mark Ajax requests.
2. The `ajaxIndicatorHeaderValue` is the value that the header must contain in order to be identified as an Ajax request. Again the default value is set to jQuery's `XMLHttpRequest`.

Per default springs `org.springframework.web.servlet.DispatcherServlet` does not handle `OPTIONS` request. You have to enable it via an `init-param` within your `web.xml`. *CoreMedia DXP 8* comes with that parameter set to `true`.



Solution 2: The Proxy

To solve this problem the classical way, the Ajax request needs to be sent to the same origin than the whole page request in step 1 was. The next image shows the solution to this problem: A reverse proxy needs to be put in front of both the CAE and the WCS.

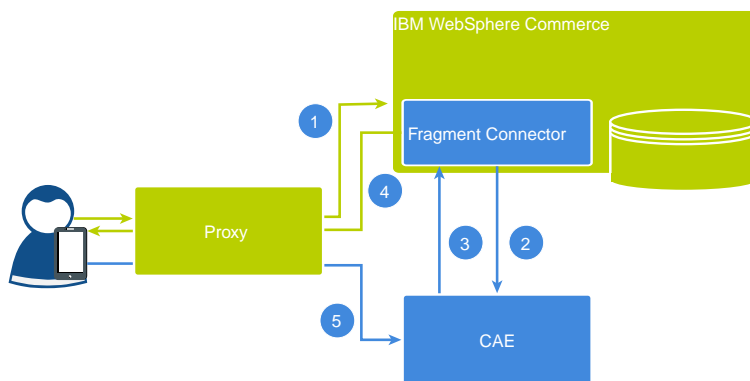


Figure 5.6. Cross Site Scripting with fragments

Actually, you may use any proxy you feel comfortable with. The following snippet shows the configuration for a Varnish. Two back ends were defined, one for the CoreMedia LiveContext CAE named `blueprint` and another one for the IBM WebSphere Commerce server named `commerce`.

The `vcl_recv` subroutine is called for every request that reaches the Varnish instance. Inside of it the request object `req` is examined that represents the current request. If its `url` property starts with `/blueprint/`, it will be sent to the CoreMedia LiveContext CAE. Any other request will be sent to the commerce system. (`~` means "contains" and the argument is a regular expression)

Now, if you request an Aurora URL through Varnish and the resulting page contains a CoreMedia LiveContext fragment including a dynamic part that must not be cached, like the sign in button, the Ajax request will work as expected.

```

    backend commerce {
        .host = "ham-its0484-v";
        .port = "80";
    }

    backend blueprint {
        .host = "ham-its0484";
        .port = "40081";
    }

    sub vcl_recv {
        if (req.url ~ "^/blueprint/") {
            set req.backend = blueprint;
        } else {
            set req.backend = commerce;
        }
    }

```

5.1.3 Extending the Shop Context in Commerce-led Integration Scenario

To render personalized or contextualized info in content areas it is important to have relevant shop context info available during CAE rendering. It will be most likely user session related info, that is available in the *IBM WCS* shop only and must now be provided to the back-end CAE. Examples are the user id of a logged in user, gender, the date he was logged in the last time or the names of target groups he belongs to, up to the info which campaign should be applied. Of course these are just examples and you can imagine much more. So it is important to have a framework in order to extend the transferred shop context information flexibly.

The relevant shop context will be transmitted to the CoreMedia CAE automatically as HTTP header parameters and can there be accessed for using it as "personalization filter". It is a big advantage of the dynamic rendering of a CoreMedia CAE that you can easily process this information at rendering time.

The transmission of the context will be done automatically. You do not have to take care of it. On the one end, at the IBM WebSphere Commerce System, there is a context provider framework where the context info is gathered, packaged and then automatically transferred to the back-end CAE. A default context provider is active and can be replaced or supplemented by our own `ContextProvider` implementation.

Implement a custom `ContextProvider`

Let's imagine you have to implement a new `ContextProvider` that will extend the shop context. It should be possible to apply it for all available content slots or only for a certain one because of its supposed exceptional specialty.

First, you write a class that implements the `ContextProvider` interface. The `ContextProvider` interface demands the implementation of a single method.

```
package com.coremedia.livecontext.connector.context;

import javax.servlet.http.HttpServletRequest;

public interface ContextProvider {

    /**
     * Add values to the given context.
     * @param contextBuilder the contextBuilder - the means to add
     * entries to the entry
     * @param request - the current request, from which e.g. the
     * session can be retrieved
     * @param environment - an environment, not further specified
     */
    void addToContext(ContextBuilder contextBuilder, HttpServletRequest
        request, Object environment);
}
```

*Example 5.1. Context-
Provider interface
method*

There can be multiple `ContextProvider` instances chained. Each `ContextProvider` enriches the Context via the `ContextBuilder`. The resulting Context wraps a map of key value pairs. Both, keys and values have to be strings. That means if you have a more complex value, like a list, it is up to you to encode and decode it on the back-end CAE side. Be aware that the parameter length can not be unlimited. Technically it is transferred via HTML headers and the size of HTML headers is limited by most HTTP servers. As a rough upper limit you should not exceed 4k bytes for all parameters. You should also note that this data must be transmitted with each back-end call.

As a rough upper limit you should not exceed 4k bytes for all parameters, as they will be transmitted via HTTP headers.



All `ContextProvider` implementations are configured via the property `com.coremedia.fragmentConnector.contextProvidersCSV` in the file `coremedia-connector.properties` as a comma separated list. The configured `ContextProvider` instances are called each time a CMS fragment is requested from the CAE back-end.

Read shop context values

On the back-end CAE side the shop context values will be automatically provided via a Context API. You can access the context values during rendering via a Java API call.

All fragment requests are processed by the `FragmentCommerceContextInterceptor` in the CAE. This interceptor calls `LiveContextContextAccessor.openAccessToContext(HttpServletRequest request)` to create and store a `Context` object in the request. You can access the `Context` object via `LiveContextContextHelper.fetchContext(HttpServletRequest request)`.

Example 5.2. Access the Shop Context in CAE via Context API

```
import com.coremedia.livecontext.fragment.links.context.Context;
import
com.coremedia.livecontext.fragment.links.context.LiveContextContextHelper;

import javax.servlet.http.HttpServletRequest;

public class FragmentAccessExample {
    ...
    private LiveContextContextAccessor fragmentContextAccessor;

    public void buildContextHttpServletRequest request(){
        fragmentContextAccessor.openAccessToContext(request);
    }

    public String getUserIdFromRequest(HttpServletRequest request){
        Context context = LiveContextContextHelper.fetchContext(request);

        return (String) context.get("wc.user.id");
    }
    ...
}
```

5.1.4 Extending with Fragments

A pure e-Commerce system is focused on the more transactional aspects of the buying process. To create a more engaging user experience you can augment the catalog pages with editorial content from the CMS. This includes, articles, images or videos.

There are two types of shop pages that can be extended by the CoreMedia CMS:

Types of augmentable pages

- ➔ **Catalog Pages** that are part of the catalog hierarchy, like a Category Overview or Landing Page and a Product Detail Page (PDP). They are extended by `Augmented Categories` in the CMS.
- ➔ **Other Pages** that are not located in the catalog hierarchy, for example, all subordinate shop pages like "Contact Us", "Log On", "Checkout", "Register" or "Search Result", which also belong to a shop but don't have a category or an product connected with. Even the homepage and other special topic pages belong to this type. These pages are extended by `Augmented Pages` in the CMS.

In addition, you can show complete CMS pages in the context of the WCS.

The basis for augmentation is the use of the CoreMedia Content Widget or the `lc:include` tag in the WCS system.

The augmentation process

On the IBM WCS side add the CoreMedia Content Widget to IBM WCS page layouts. Here, content from the CMS will be shown. The value of the `placement` property connects the widget with a placement in a CMS layout. Technically, the widgets use the `lc:include` tag. See [Section “CoreMedia Widgets” \[184\]](#) and [Section “The CoreMedia Include Tag” \[186\]](#) for details.

When you have added the widget and you have connected the WCS and CMS systems, you can augment WCS content in *Studio*. [Section 5.10.5, “Augmenting WCS Content” \[231\]](#) describes the required steps.

CoreMedia Widgets

On the *IBM WCS* side it is necessary to define slots where the CMS content can be displayed. This is normally done by adding the CoreMedia Content Widgets to an *IBM WCS* page layout.

In other cases, where a widget cannot be used, it can also be achieved by directly adding an `lc:include` tag into a JSP within the *IBM WCS* workspace. This is typically done in advance during the project phase. Later, editors will only deal with `Augmented Categories` and `Augmented Pages` that they can edit and preview via *CoreMedia Studio*.

The content that is shown in the *CoreMedia Content Widget* is taken from a placement in the augmented content item, whose name corresponds with the name set in the widget. See [Figure 5.7, “Connection via placement name” \[184\]](#) for an example. Note, that the name of the placement shown in the Studio form is only a localized label. The name in the *Content Widget* must match with the technical name in the page grid definition. If the widget defines no placement, the full page grid is taken.

Adding the CoreMedia Content Widget

Using the lc:include tag

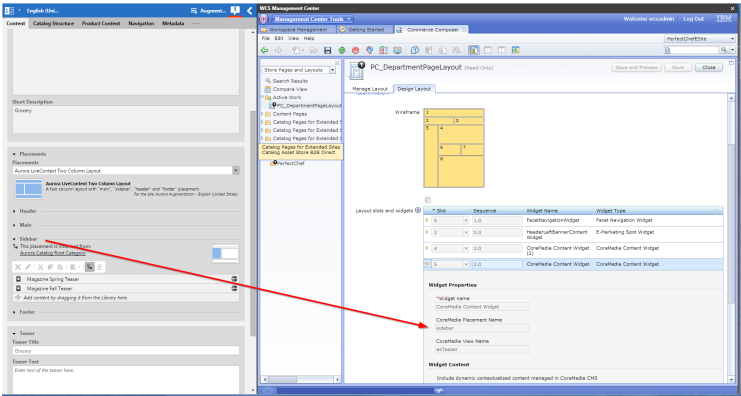


Figure 5.7. Connection via placement name

The CoreMedia widgets are *IBM Commerce Composer Widgets* that display content or assets from the CMS on any page managed through the *IBM Commerce Composer*. After the CoreMedia widgets have been deployed on the WCS side (see [Section](#)

3.4.11, “Deploying the CoreMedia Widgets” [73]), two CoreMedia widgets are available in the *IBM Commerce Composer*:

- CoreMedia Content Widget
- CoreMedia Asset Widget

Figure 5.8. CoreMedia Widgets in Commerce Composer



Technically, the CoreMedia Widgets use the `lc:include`. See [Section “The CoreMedia Include Tag” \[186\]](#) for a description.

The CoreMedia Content Widget

You can use the *Content Widget* like any other *Commerce Composer Widget*. It has the following configuration options:

Option	Description
Widget name	The widget name.

Table 5.1. CoreMedia Content Widget configuration options

Option	Description
CoreMedia Placement Name	The name of the placement as defined in CoreMedia CMS. Content on page grids in CoreMedia are defined through so called placements. Each placement is associated with a specific position of the page grid through its name. Using CoreMedia Studio the editor can add content to the placement which will be shown at the associated position of the page grid and subsequently in the layout of this CoreMedia Content Widget.
CoreMedia View Name	The view of the placement as defined in CoreMedia CMS. Each placement can be rendered with a specific view which needs to be predefined to handle the content in a placement.

The CoreMedia Product Asset Widget

The Product Asset Widget is part of the *CoreMedia Advanced Asset Management* module described in [Section 8.7, “Advanced Asset Management” \[443\]](#). This module requires a separate licence.



You can use the *CoreMedia Product Asset Widget* like any other *Commerce Composer* Widget. It has the following configuration option:

Option	Description
Display Pictures and Videos	If checked, a picture gallery is rendered from CMS pictures and videos that are associated with the product.
Orientation	The orientation of the pictures (only relevant if pictures are included). The possible values are <i>Square</i> and <i>Portrait</i>
Include Downloads	If checked, an <i>Additional Downloads</i> list is rendered from CMS <i>Download</i> documents that are associated with the product.

Table 5.2. CoreMedia Product Asset Widget configuration options

The CoreMedia Include Tag

Behind the scenes of the CoreMedia Commerce Composer Widget ([Section 3.4.11, “Deploying the CoreMedia Widgets” \[73\]](#)) works the CoreMedia `include` tag, which you may use in your own JSP templates to embed CoreMedia content on the WCS side. In general it is used like this:

```
<%@ taglib prefix="lc"
uri="http://www.coremedia.com/2014/livecontext-2" %>
<lc:include
    storeId="${WCPParam.storeId}"
```

```
locale="${WCPParam.locale}"
productId="${WCPParam.productId}"
categoryId="${WCPParam.categoryId}"
placement="${param.placement}"
view="${param.view}"
externalRef="${WCPParam.externalRef}"
exposeErrors="${not empty WCPParam.externalRef
    && empty WCPParam.categoryId
    && empty WCPParam.categoryId}"
httpStatusVar="fragmentHttpStatus"/>
```

All parameters are described in the next two sections.

Include Tag Reference

The tag attributes have the following meaning:

Parameter	Description
<i>storeId, locale</i>	These attributes are mandatory. They are used in the CAE to identify the site, which provides the requested fragment.
<i>productId, categoryId</i>	These attributes are used in the CAE to find the context which will be used for rendering the requested fragment. Both parameters should not be set at the same time since depending on the attributes set for the include tag, different handlers are invoked: If the <i>categoryId</i> is set, <i>CategoryFragmentHandler</i> will be used to generate the fragment HTML. If the <i>productId</i> is set, <i>ProductFragmentHandler</i> will be used to generate the fragment HTML.
<i>pageId</i>	This parameter is optional. Usually, the page ID is computed from the requested URL (the last token in the URL path without a file extension). If you set the parameter, the automatically generated value is overwritten. On the blueprint side an <i>Augmented Page</i> will be retrieved to serve the fragment HTML. The transmitted page ID parameter must match the <i>External Page ID</i> of the <i>Augmented Page</i> . You might use the parameter, for example, in order to have one <i>CoreMedia</i> page to deliver the same content to different <i>WCS</i> pages.
<i>placement</i>	This attribute defines the name of a placement in the page grid of the requested context. In the example for the header fragment, the "header" placement was used. If you do not want to render a certain placement but a view of the whole context (generally a <i>CMChannel</i>), you may omit it. If the view attribute isn't set, the "main" placement will be used as default instead. This attribute can be combined with the <i>externalRef</i> attribute. In this case the placement will be rendered for a specific <i>CMChannel</i> , so the external reference must point to a <i>CMChannel</i> instance.

Table 5.3. Attribute of the Include tag

Parameter	Description
<i>view</i>	The attribute "view" defines the name of the view which will render the fragment. For including the CoreMedia CSS links into the header, chose the "css" view. And because no placement was provided, the CAE was told to render the "css" view of the whole root channel. If you omit the view, the default view will be used.
<i>externalRef</i>	This attribute is used in the CAE to find content. Several formats are supported here as described in the previous section. The attribute can be used in combination with the "view" and/or "parameter" attribute.
<i>parameter</i>	This attribute is optional and may be used to apply a request attribute to the CAE request. The request attribute is stored using the constant <code>FragmentPageHandler.PARAMETER_REQUEST_ATTRIBUTE</code> . The value may be read from a triggered web flow, for example, to pass a redirect URL back to the commerce system once the flow is finished. The attribute also supports values to be passed in JSON format (using single quotes only), for example <code>parameter="{ 'test': 'some value', 'value': 123 } "</code> . The key/values pairs are available in the <code>FragmentParameters</code> object and may be accessed using the <code>getParameterValue(String key)</code> method. Other additional values, like information about the current user that should be passed for every request, may be added to the request context that is build when the commerce system requests the fragment information from the CAE (see next section).
<i>var</i>	This attribute is optional. If set, the parsed output of the CAE is not written in the parsed output stream but in a page attribute named like the <i>var</i> parameter value. This allows you, for example, to replace or transform parts of the CAE result or, if empty, to render a different output.
<i>exposeErrors</i>	This attribute is optional. If set to true, the tag will expose any errors that occur during the interaction with the CMS. These errors are then directly written to the response. Thus, the <i>IBM WCS</i> has the ability to handle the errors, e.g. to show an error page.
<i>HttpStatusVar</i>	This attribute is optional. If set, the http status code of the fragment request is set into a page attribute named like the <i>HttpStatusVar</i> parameter value. This allows you, for example, to react on the result code, e.g. set the fragment as un-cacheable in IBM Dynacache.

External References

Any linkable CoreMedia content can be included as fragment by specifying a value for the `externalRef` attribute. The value of the attribute is applied to the first `ExternalReferenceResolver` predicate that is applicable for the `externalRef` value. The Spring list `externalReferenceResolvers` which contains the supported `ExternalReferenceResolvers` is injected to the `ExternalRefFragmentHandler`. This section shows the supported formats that are applicable for the existing resolvers.

The following table shows an overview about the possible values for the `externalRef` attribute.

Value Type	Example	Description
Content ID	cm-coremedia:///cap/content/4711	Includes the content with the given cap id as fragment. The root channel of the corresponding site will be used as context.
Numeric Content ID	cm-4712	Works the same way like the cap id include, only with the numeric content ID.
Absolute Content Path	cm-path!!Themes!basic!img!icons!ico_rte_link.png	Includes the content with the given absolute path. All exclamation marks (!) after the prefix 'cm-path!' will be mapped to slashes (/) to provide a valid absolute CMS path. The given path may not contain 'Sites' (referencing content of a different site is not allowed). The <i>storeId</i> and <i>locale</i> parameter are still mandatory for this case.
Relative Content Path	cm-path!actions!Login	Includes the content with the given path treated as a relative path from the site's root folder. All exclamation marks (!) after the prefix 'cm-path!' will be mapped to slashes (/) to provide a valid relative CMS path. The given path may not contain '..' (going up in the hierarchy). The site is determined through the <i>storeId</i> and <i>locale</i> parameter.
Numeric Context and Content ID	cm-3456-6780	The prefix is the numeric content ID of the context to be rendered. The suffix is the numeric content ID of

Table 5.4. Supported usages of the `externalRef` attribute

Value Type	Example	Description
		the content to be rendered with the given context.
Segment Path	cm-segmentpath:!perfectchef!on-the-table	<p>The actual value (excl. the format prefix <code>cm-segmentpath:</code>) denotes a segment sequence, separated by exclamation marks. The segments are matched against the values of the <code>segment</code> properties of the content. The very last segment denotes the actual content. The other segments denote the navigation hierarchy which determines the context of the content. The example value references a linkable content with the segment <code>on-the-table</code> in the context of a channel <code>perfectchef</code> (which is apparently the root channel, since it consists of a single segment). The context and the content must fulfill the Blueprint's context relationship, otherwise the request is handled as invalid.</p> <p>Segment Path external references are resolved by querying the Solr search engine. The <i>CAE Feeder</i> must be running for up-to-date results.</p>
Search Term	cm-searchterm:summer	<p>Includes the content that contains the given search term (specified after the prefix <code>cm-searchterm:</code>). This resolver is typically used to resolve search landing pages. By default, contents of type <code>CMChannel</code> below the segment path <code><root segment>/livecontext-search-landing-pages</code> are checked if their <code>keywords</code> search engine index field contains the term. Matching is case-insensitive by default and can be customized by using a different search engine field or field type. The value of the segment path which is used to identify the SLP channel is configured with</p>

Value Type	Example	Description
		<p>the property <code>liveon</code> <code>text.slp.segmentPath</code> in file <code>component-lc-ecommerce-</code> <code>ibm.properties</code>.</p> <p>Content type and search engine field can be configured with Spring properties <code>searchTermExternalReferenceResolver.contentType</code> and <code>searchTermExternalReferenceResolver.field</code>, respectively. The segment path is configured as relative path after the root segment. The configured segment path value must not start with a slash.</p> <p>Search term lookup is cached, by default for 60 seconds. You can configure the cache time in seconds with Spring property <code>searchTermExternalReferenceResolver.cacheTime</code> and the maximum number of cached search term lookups with <code>searchTermExternalReferenceResolver.cacheCapacity</code> (defaults to 10000).</p> <p>Search Term external references are resolved by querying the Solr search engine. The <i>CAE Feeder</i> must be running for up-to-date results.</p>
Metadata Retrieval	cm-metadata	<p>A <code>cm:include</code> tag that uses this external reference format is used in the <code>Metadata.jsp</code> which is part of the commerce workspace. The JSP is included in the head section of several commerce templates. It is used to retrieve data for the HTML tags <code>title</code> and <code>meta</code>. The CAE will render the corresponding HTML using the <code>Navigation.metadata.ftl</code> template.</p>

Finding Handlers

You can control the behavior of the `include` tag by providing different sets of attributes. Depending on the used attributes, different handlers are invoked to generate the HTML.

The `CoreMedia lc:include` tag requests data from the CAE via HTTP. Each attribute value of the `include` tag is passed as path or matrix parameter to the `FragmentPageHandler`. In order to find the matching handler, the `FragmentPageHandler` class calls the `include` method of all fragment handler classes defined in the file `livecontext-fragment.xml`. The first handler that returns "true" generates the HTML. [Example 5.3, "Default fragment handler order" \[192\]](#) shows the default order:

Example 5.3. Default fragment handler order

```
<util:list id="fragmentHandlers"
value-type="com.coremedia.livecontext.fragment.FragmentHandler">
  <description>This list contains all handlers that are used for
fragment calls.</description>
  <ref bean="externalRefFragmentHandler" />
  <ref bean="externalPageFragmentHandler" />
  <ref bean="productFragmentHandler" />
  <ref bean="categoryFragmentHandler" />
</util:list>
```

If the handlers are in the default order, then the table shows which handler is used depending on the set attributes. An "x" means that the attribute is set, a "-" means that the attribute is not allowed to be set and no entry means that it does not matter if something is set. For more details, have a look into the handler classes.

Table 5.5. Fragment handler usage

External Reference	Page ID	Category ID	Product ID	Used Handler
x				ExternalRefFragmentHandler
-	x	-	-	ExternalPageFragmentHandler
-			x	ProductFragmentHandler
-		x	-	CategoryFragmentHandler

Fragment Request Context

In addition to the passed request parameters, a context is build by the registered `ContextProvider` implementations that are part of the commerce workspace. The context provider passes context information as header attributes to the CAE. For more details see [Section 5.1.3, "Extending the Shop Context in Commerce-led Integration Scenario" \[181\]](#).

CMS Error Handling

Since the CoreMedia `include` tag requests data from the CAE via HTTP, errors can occur. The error handling can be controlled by different parameters. If the `com.coremedia.fragmentConnector.isDevelopment` property (see [Section 3.4.9, “Deploying the CoreMedia Fragment Connector” \[68\]](#)) is set to `true`, the `include` tag will embed occurring error messages as strings into the page output. You may not want to see such information on the live side, thus the flag can be set to `false` and all output will be suppressed (the errors are only visible in the log).

This behavior is sufficient for providing additional (possibly optional) information on a page, a banner or teaser, for instance. But if the requested content is the major content of a page, then it is not desirable to deliver a mainly empty page. In such a case the IBM WCS should be able to handle the error situation and answer in an appropriate form. That could be e.g. an 404 error page.

For this purpose the `exposeErrors` parameter was introduced for the `include` tag. If this parameter is set to `true`, the tag will expose any error that occur during the interaction with the CMS. These errors are directly written to the response. Sending a response with an error status code (e.g. 404) requires that still nothing has been written to the `Response` object. Therefore this flag should only be set on the `include` tag if rendered early enough before any other response code has been set.

In our IBM WCS reference workspace the usage of the `exposeErrors` parameter is demonstrated in the `CommonJSToInclude.jspf` template. The template is executed on every page request and renders, amongst other things, the HTML head section of a page. The first occurrence of the `include` tag is used to do the error handling.

Since the template is executed for all shop pages the flag must be set depending on the target page. If it's a content centered page (it has, for example, a `cm` parameter), then the parameter would be set to `true`, in case of an category or product detail page probably not.

```
exposeErrors="${not empty WCPParam.externalRef && empty
WCPParam.categoryId && empty WCPParam.categoryId}"
```

Another possibility to handle failed fragment requests is the usage of the `http-StatusVar` parameter. If this parameter is set, the `include` tag will write the http status code of the fragment request into a JSP attribute/variable. You can then add JSP code to react on specific result codes and for example disable caching of this fragment in IBM Dynacache.

```
<lc:include ...
  httpStatusVar="status"/>
```

```
...  
<c:if test="${not empty status && status >= 400}">  
  ... // error handling  
</c:if>
```

5.2 Content-led Integration

In the content-led scenario, WCS and CMS system are equal partners. It is possible, that the CoreMedia CAE delivers all content to the customer, while augmenting the pages with content, such as prices, from the WCS.

- ➔ [Section 5.2.1, “Content-led Integration Overview” \[195\]](#) gives an overview over the request flow in the content-led scenario.
- ➔ [Section 5.2.2, “Status Synchronization in the Content-led Integration Scenario” \[196\]](#) describes how the user state is synchronized between the WCS and CMS systems. This includes authorization, password reset and more.
- ➔ [Section 5.2.3, “Configuring Protocol-less Links for WCS” \[202\]](#) describes how you configure protocol-less links required for the content-led scenario.

5.2.1 Content-led Integration Overview

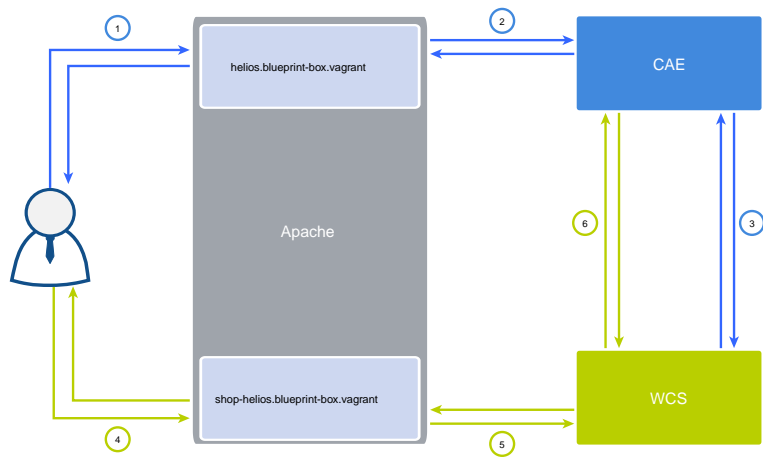


Figure 5.9. Content-led integration scenario

The most obvious difference to the commerce-led scenario in the content-led scenario is the presence of a second virtual host, that separates both systems, the CAE and the commerce system, clearly from one another. Here the CAE is the fully equal partner of the commerce system with the potential to become the driving force for rendering the whole front end.

The description of a typical request flow through the system, as shown in [Figure 5.9, “Content-led integration scenario” \[195\]](#), clarifies the different roles of the CAE and the commerce system in this scenario.

1. The user requests a marketing driven landing page of a shop system.

2. The virtual host for the CAE forwards the request to the CAE.
3. Part of the requested page are various product teasers, with dynamic prices. Hence, the CAE needs to fetch corresponding information from the commerce system.
4. After receiving the page from the CAE, the user decides to click on a product teaser to see the corresponding product details. The link, rendered by the CAE as part of the landing page, directs the user to the virtual host of the commerce system.
5. The virtual host forwards the request to the commerce server.
6. As the requested Product Detail Page (PDP) contains a CoreMedia fragment, the WCS requests it from the CAE and sends the whole PDP back to the user.

From the example follows, that the commerce-led integration scenario described in [Section 5.1, “Commerce-led Integration Scenario” \[176\]](#) is a subset of the content-led scenario. The request flow 4->5->6 uses the exact same technique to handle included CoreMedia fragments into WCS pages as described in the commerce-led scenario. The only difference is that resources or dynamic fragments fetched via Ajax requests are not handled by the virtual host of the commerce system. Instead, they are sent to the CAEs virtual host.

5.2.2 Status Synchronization in the Content-led Integration Scenario

Take a look at figure [Figure 5.9, “Content-led integration scenario” \[195\]](#). As you can see, the CAE and the commerce system stand side by side as equal partners from a users point of view. A user is allowed to request pages from both systems at any given time.

Motivation

This architecture forces the CAE to synchronize any user sessions on the commerce system with its own. A user that browses the CAE and afterwards visits the WCS must keep his session and vice versa a user browsing the WCS going to the CAE afterwards must keep his state as well.

This section describes how the synchronization of this state is implemented by the CoreMedia CAE.

What Is The Users State?

IBM WebSphere Commerce represents the state of a user session using cookies. To understand the synchronization of a users state across both systems you need to understand how those cookies may flow through the system. Take a closer look at [Figure 5.10, “Content-led integration scenario with cookies” \[197\]](#). In addition to the request flow, the dashed green and blue arrows represent the flow of cookies.

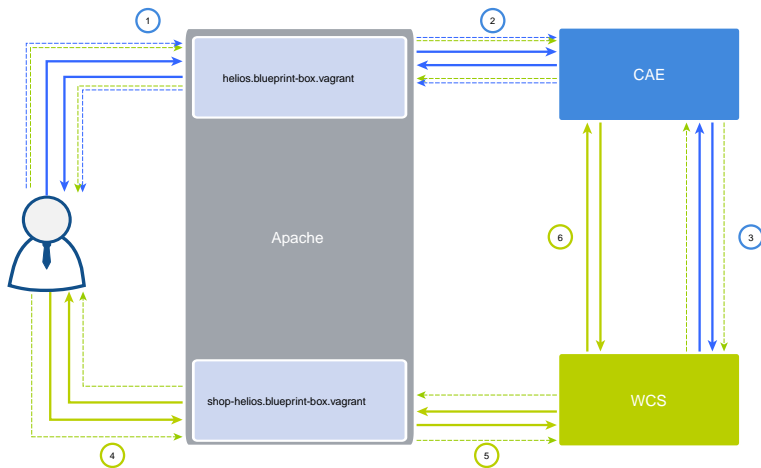


Figure 5.10. Content-led integration scenario with cookies

You can see that cookies may flow nearly everywhere. No matter where a request starts and where it ends, either between the browser and the CAE or between the CAE and the WCS, every node may be the source as well as the receiver of cookies.

Two things that need explanation. First, two kinds of cookies flow from the browser to the CAE, cookies which were originally created in the commerce system and cookies that are created by the CAE. This is necessary because the CAE must send the commerce cookies to the commerce system as part of its back-end calls. Second, for fragment requests (labeled with 6), no CoreMedia cookies are needed, hence, the browser does not need to send the CAE cookies to the commerce server.

Hence, CoreMedia had to answer the following questions:

- How can the CAE render fragments without the need to receive its own cookies?
- How can the user's browser be enabled to send cookies received from the commerce system to the CAE?
- How can the CAE be seen as the source of WCS cookies without actually creating them by itself?
- How can the CAE synchronize its own user session with the one of the commerce system?

The following section will answer all four questions.

How does the CAE renders fragments without its own cookies?

Cookies are used for dynamic HTML snippets, which are snippets that cannot be cached because they contain user specific content. Fragments that the CAE delivers

to the commerce server should never include such dynamic HTML snippets because this would prevent a CDN or other caching infrastructure from caching complete WCS pages.

But if the CAE does only return static HTML within the fragment responses to the commerce system, it does not need any cookies. Everything that needs cookies in a fragment must instead be implemented by using dynamic fragments, explained in [Section 8.2.1, “Using Dynamic Fragments in HTML Responses” \[414\]](#).

For solving the same origin policy problem that would occur for dynamic (Ajax) requests against the CAE, that originated in a static CAE fragment delivered via the WCS, CoreMedia provides a solution described in [Section 5.1.2, “Solutions for Same-Origin Policy Problem” \[177\]](#)

How does the browser deliver WCS cookies to the CAE?

The browser sends cookies to a server that runs in the same domain, that is saved with the cookie. In general the cookie domain of a cookie is left empty, so that the browser stores the exact host name of the server that responded to a request. But because the CAE and the commerce system must have different host names (via their virtual host), the CAE would never receive WCS cookies.

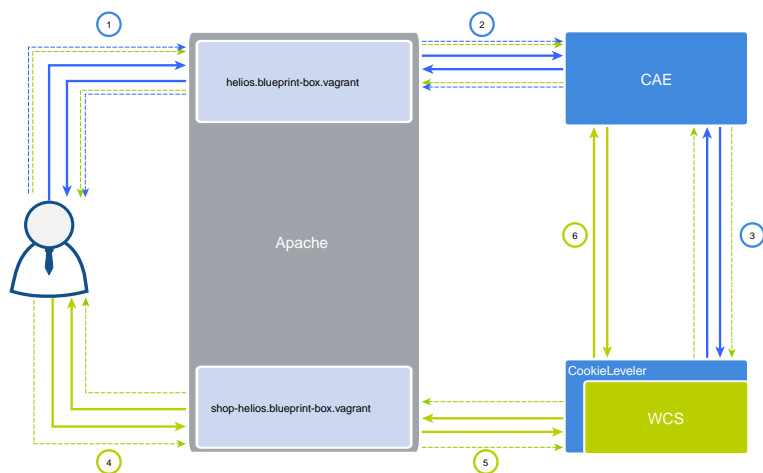


Figure 5.11. Content-led integration scenario

The solution to this problem is fairly simple. A servlet filter, the so called cookie leveler, runs in front of any WCS storefront call. It wraps the `HttpServletResponse` into a custom one, that intercepts `addCookie()` method calls in order to set the cookie domain to a configurable value.

You have to enable the cookie leveler from within your `web.xml` file of your storefront and preview webapp, which is described in [Section 3.4.5, “Configuring the Cookie Domain” \[65\]](#)

The cookie leveler should be executed prior to any other filter that may add cookies to the response. In general CoreMedia recommends you to put its filter mapping definition in front of any other filter mapping.

There is one cookie that cannot be customized that way, the `JSESSION` cookie, which is set by the WebSphere servlet container. You have to configure it via the usual mechanisms provided by IBM, for example via the IBM console.

Now the CAE and the WCS only need to be put into the same domain, for example `helios.blueprint-box.vagrant` for the CAE and `shop-helios.blueprint-box.vagrant` for the WCS. The cookie domain must then be configured to be `.blueprint-box.vagrant`

The cookie domain must not be a top level domain, for example `.com`, because that would mean, every website in the `.com` domain will receive the cookies. Because that does not make any sense, cookies with only a top level domain are generally not sent at all.



The CAE as WCS Cookie Source

From the point of view of the user, the CAE seems to be a second source for WCS cookies as follows: A user requests a landing page from the CAE. Part of this page is a login link. The login will be again handled by the CAE. But to authenticate the user the CAE delegates to the WCS (call 3 in the figure). That is a store front call and WCS creates the usual WCS cookies as result of a successful authentication. The CAE receives them and copies them to the original HTTP response that is sent back to the user.

The next figure shows the copy process in the top right corner. Moreover, you can see that any WCS cookies, that were sent to the CAE will be copied to any back-end call to the WCS.

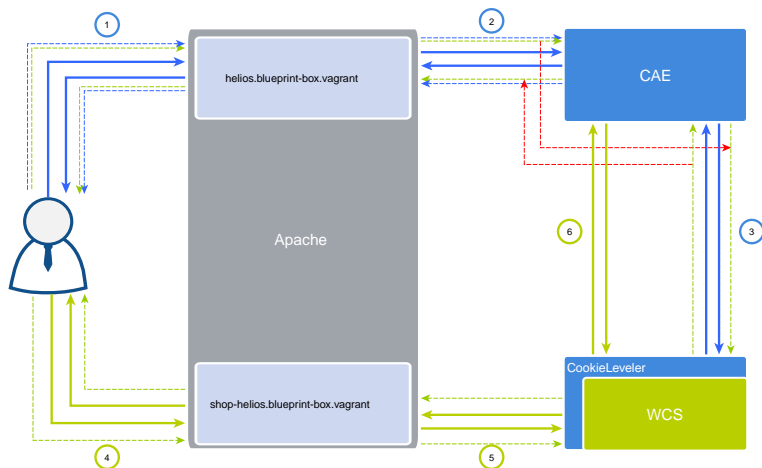


Figure 5.12. Content-led integration scenario

How are sessions between CAE and WCS are synchronized?

Session synchronization is done by the CAE only. The commerce system does not know anything about the CAE. Although it is the CAE that executes the session synchronization, it is the WCS that is the leading system for doing so.

There are two states to look at, the `org.springframework.security.core.context.SecurityContext` that reflects the authentication state remembered within the CAE and the authentication state within the WCS.

Synchronization is done by the `com.coremedia.livecontext.services.SessionSynchronizer`, which is triggered by the `com.coremedia.livecontext.handler.SessionSynchronizationInterceptor`, a Spring Handler Interceptor that intercepts dynamic fragment requests. If the current user is logged into the CAE but not into the WCS, the user will be logged out from the CAE. And if the user is not logged into the CAE but into WCS, he will be logged into the CAE.

Authentication Process

CoreMedia's solution for this is a `UserService` which is triggered by the authentication forms. The interface `UserService` describes operations for authentication and for changing the password. Part of the delivery is a concrete `UserService` `UserServiceImpl` which tries the authentication operation against the third-party commerce system. For IBM WebSphere the `WcsUserServiceImpl` is an implementation of the interface `CommerceUserService`. An instance for this is provided as a Spring bean with the id `liveContextUserService`.

The authentication is realized through a Spring security manager that is configured in file `livecontext-cae-spring-security.xml`. The Spring security configuration of *CoreMedia DXP 8* provides an `AuthenticationProvider` that is implemented in the class `LiveContextUserAuthenticationProvider`. The provider looks up the Elastic Social user for a given user name or email address. If the user exists, the credentials are checked using login method of the `CommerceUserService`.

Registration

When a user registers at the system, the `UserServiceImpl.registerUser()` method is called. If the user data is valid, the user is registered in the CMS first. The registration call against the IBM WebSphere Commerce system is executed afterwards. If both calls are successful, the user is notified about a successful registration. If the IBM WebSphere Commerce registration fails, the user is deleted from the CMS afterwards.

The registration call of the `UserServiceImpl` sets the system property `elastic.automatic.user.activation` to "true". This ensures that the user is automatically a registered user in the CMS and no separate registration confirmation has to be executed. Also, for security reason the password of the user is only stored in the IBM WebSphere Commerce server and reset to an empty string before it is stored in the CMS. Since the authentication is executed against the IBM WebSphere Commerce system, it is not used here.

The field mapping between the Elastic Social and commerce user models can be customized in the class `PersonMapper`. The class implements the mandatory mappings that are executed during the registration and the profile editing of the user.



Edit User Details

Authenticated users can update their details and their passwords in the profile settings. All data is stored in the commerce system. Only the user name and password (if editable) are synchronized between the IBM WebSphere system and the CMS. To edit the user data the user detail model of *Elastic Social* is used by the specialized class `LiveContextUserDetails`. Once the user submits the profile form, the data of this model is applied to the `Person` model of the e-Commerce API and stored in the commerce system.

Password Reset

If users have forgotten their password, an email or some other type of notification is sent via the IBM WebSphere Commerce server. The message contains the newly generated password. The user can login on the store again and update the password in the profile settings.

The password reset is executed by a custom REST service handler `PasswordResetHandler` that has to be installed for the Commerce system. The update password method of the handler will reset the password for unauthorized users and update the password for authenticated users. Once the password is reset/updated, an email will be sent by the Commerce system. Ensure that an SMTP server is configured properly in the IBM WebSphere Administration Console for that. Also, the Administration Console allows inspecting the mail queue of pending mails (if the SMTP server has not been setup yet).

The default password reset behavior differs from the default one that has been implemented for IBM's Aurora store. If unauthorized users reset their passwords, they can not login until the generated password has been updated to a new one. Every link in the store points to the update password form.

This behavior can not be disabled in the developer edition of the IBM WebSphere Commerce server, but should be disabled for the production environment: The default login flow that is configured in the XML file `lc/lc-cae/src/main/resources/com/coremedia/livecontext/es/web-flow/com.coremedia.blueprint.elastic.social.cae.flows.Login.xml` will try to log out users if they authenticate against the IBM WebSphere Commerce system with an expired password. This log out call will fail since the system assumes that the user has to be logged in until the password has been updated. On the CAE site, this will result in an inconsistent cookie state.



Error Handling

Since all user data is stored in the IBM WebSphere Commerce server and the user authenticates against the server too the catalog API provides error handling so that errors thrown by the IBM WebSphere Commerce system can be handled by the CAE. IBM WebSphere Commerce provides several error messages with error codes that may occur during the registration, authentication or password reset/update of the user. The error codes are automatically extracted by the exception `CommerceRemoteException`. The Elastic Social web flow looks up commerce errors using the resource key format `commerce.error.<ERROR_CODE>`. The current error mapping can be configured in the sample content files in `/test-data/content/Settings/Options/Bundles/Livecontext*`.

5.2.3 Configuring Protocol-less Links for WCS

In the content-led scenario, commerce must render protocol-less links (without "http:" or "https:" prefix), so that during a protocol change links of included CAE fragments will use the correct protocol automatically. The configuration for this can be set in the `wc-server.xml` of the WCS:

```
<SEOConfiguration defaultUrl="" dynamicUrl="true" enable="true"  
constructUrlWithoutHost="true"/>  
<UrlTagConfiguration useRelativePath="true"/>
```

5.3 Communication

CoreMedia DXP 8 and IBM WebSphere communicate over REST interfaces. The concrete communication differs slightly based on the selected deployment scenario which are the *content-led scenario* and the *commerce-led scenario*. The following picture shows the communication how fragments of a page are delivered from the CAE and from the commerce system.

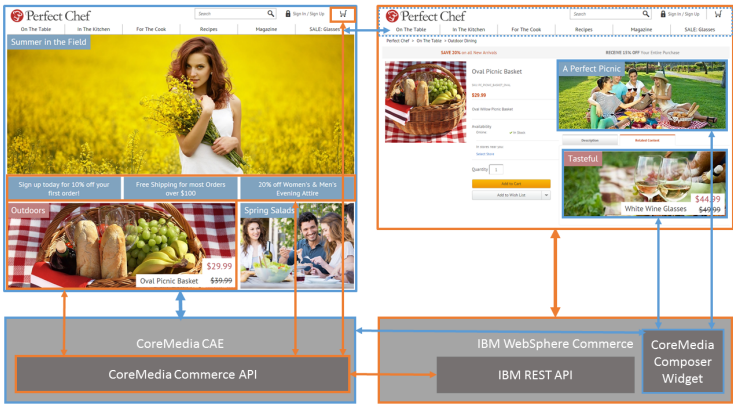


Figure 5.13. Content-led/Commerce-led scenario communication

In the *content-led scenario* common pages and detail pages are delivered by the CAE, including fragments like the login information and the shopping cart. For product teasers and the shopping cart, additional data is requested from the commerce system to display the price and other details. The CAE can only render products as teaser. The document type `CMProductTeaser` is used for this, because it contains a reference to the concrete product variant inside the commerce system. Therefore, also the teaser text, title and picture are read from the CMS and only the pricing information is read (through the commerce API) from the product:

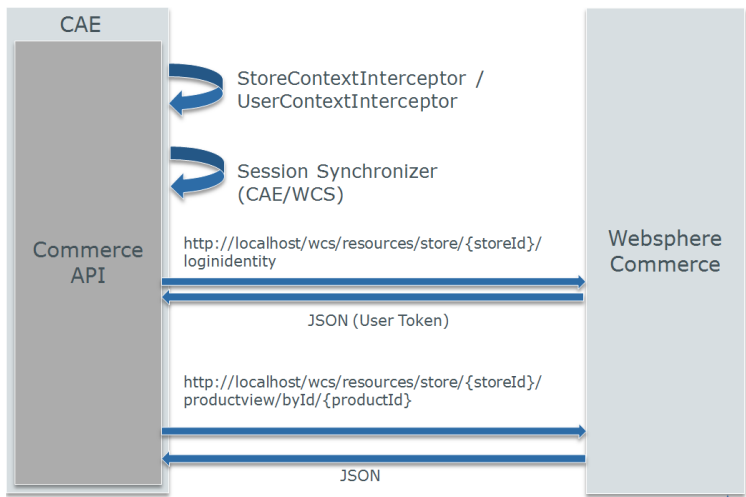


Figure 5.14. Example of a Commerce API Re-request

The image shows what happens when the CAE requests data from commerce. First of all, the `StoreContextInterceptor` and `UserContextInterceptor` are executed to determine the store (and locale) and user the request is executed for. Depending on the commerce request, an additional authentication is performed to authorize the user to request the data. The response authentication token is used to retrieve the data itself, for example, the details of a product or a shop category. The return format is JSON which is wrapped into Java objects. These objects are used to render commerce data like the name of a product inside a product teaser.

When the user clicks on a product teaser, the CAE redirects the user to the commerce system which renders the product detail page. Depending on the scenario (Content-led/Commerce-led) the header and footer fragments are rendered by the CAE or the commerce system. The commerce system requests the fragments using the *CoreMedia Fragment Connector*.

```
<lc:include storeId="${storeId}" placement="header" locale="${locale}"
categoryId="${categoryId}" />
```

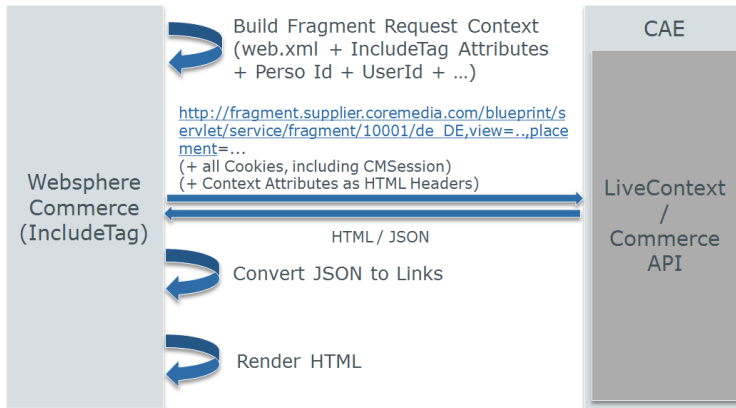


Figure 5.15. Example of a Fragment Connect- or Request

Fragment requests are executed through the `lc:include` tag. The tag is parameterized with several attributes and additional HTTP headers that are set through request context providers which are defined in the `web.xml` of the commerce system. The fragment request contains all cookies of the commerce system too. This is mandatory since the CAE needs to know if the user is already authenticated against the system to deliver the corresponding fragments.

Each response of the CAE contains HTML with JSON placeholders (wrapped inside an HTML comment). The link building is executed by the commerce system by applying a template for every link, depending on the JSON data that was returned for the link. The lookup folder for these templates are also defined in the `web.xml` of the commerce system.

5.4 Connecting with an IBM WCS Shop

To connect your *Blueprint* web applications with an IBM Commerce Store, you configure a connection on the CMS side. The connection to the IBM WebSphere Commerce system contains two parts:

- Spring configuration in the web application
- Settings configuration in *Studio* which refers to the Spring based configuration with the possibility to overwrite individual properties)

Prerequisite

Before you can connect the CoreMedia system with the WCS you need to deploy the CoreMedia extensions into your WCS system as described in [Section 3.4, “Customizing IBM WebSphere Commerce” \[58\]](#).



Spring Configuration

The information on how to connect to the *IBM WCS* system is configured in the filesystem of the web applications that use the *e-Commerce API*. These web applications are at least the Studio web application (workspace module `studio-webapp`) and the CAE applications for delivery (workspace modules `cae-preview-webapp` and `cae-live-webapp`).

The `application.properties` file below the `WEB-INF` directory of your web application contains the *IBM WCS* related configuration properties. [Example 5.4, “IBM WCS configuration in application.properties” \[207\]](#) shows the relevant parts of the file. The meaning of the values will be explained in the table below.

```
#####
# CoreMedia LiveContext Configuration
#####

livecontext.ibm.wcs.url= http://wcs-server.yourdomain.com
livecontext.ibm.wcs.secureUrl= https://wcs-server.yourdomain.com
livecontext.ibm.wcs.rest.url=
${livecontext.ibm.wcs.url}/wcs/resources/store
livecontext.ibm.wcs.rest.secureUrl=
${livecontext.ibm.wcs.secureUrl}/wcs/resources/store
livecontext.ibm.wcs.store.url=
${livecontext.ibm.wcs.url}/wcsstore/Aurora
livecontext.service.credentials.username= <wcs_username>
livecontext.service.credentials.password= <wcs_password>
livecontext.managementtool.web.url=
${livecontext.ibm.wcs.secureUrl}/lobtools/cmc/ManagementCenterMain
```

Example 5.4. IBM WCS configuration in application.properties

Table 5.6. Properties for WCS connection

livecontext.ibm.wcs.url	
Description	The general WCS URL
Example	http://wcs-server.yourdomain.com
livecontext.ibm.wcs.secureUrl	
Description	The secure WCS URL
Example	https://wcs-server.yourdomain.com
livecontext.ibm.wcs.rest.url	
Description	How to reach the WCS REST API via HTTP
Example	\${livecontext.ibm.wcs.url}/wcs/resources/store
livecontext.ibm.wcs.rest.secureUrl	
Description	How to reach the WCS REST API via HTTPS
Example	\${livecontext.ibm.wcs.secureUrl}/wcs/resources/store
livecontext.ibm.wcs.store.url	
Description	Another WCS URL to get shop resources
Example	\${livecontext.ibm.wcs.url}/wcsstore/Aurora
livecontext.service.credentials.username	
Description	The service user used to login into WCS
Example	cmsadmin
livecontext.service.credentials.password	
Description	Password of the service user
Example	changeme
livecontext.managementtool.web.url	
Description	The web URL of the commerce system's management tool
Example	\${livecontext.ibm.wcs.secureUrl}/lobtools/cmc/ManagementCenterMain

The exact store configuration depends on your store configuration in your IBM WCS environment. The store specific properties that logically define a shop instance can also be part of the Spring configuration. The following listing gives an example.

```
<util:map id="myStoreConfig">
  <entry key="store.id" value="${my.wcs.store.id}"/>
  <entry key="store.name" value="${my.wcs.store.name}"/>
  <entry key="catalog.id" value="${my.wcs.catalog.id}"/>
  <entry key="currency" value="${my.wcs.store.currency}"/>
  <entry key="dynamicPricing.enabled"
value="${my.wcs.store.dynamicPricing.enabled}"/>
</util:map>
```



```
<customize:append id="ibmStoreConfigurationsCustomizer"
bean="ibmStoreConfigurations">
  <map>
    <entry key="myStore" value-ref="myStoreConfig"/>
  </map>
</customize:append>
```

The example makes use of Spring placeholder tokens that are mapped in the properties file mentioned above. It also shows how to add the custom store configuration to the existing `ibmStoreConfigurations` map. Later this store configuration (`myStore`) can be referenced from the site configuration settings within the content. Individual values can be overwritten in the content again.

Alternatively to a `catalog.id` it is also possible to use the catalog name within a `catalog.name` property instead. It will be mapped to the current `catalog.id` at runtime. When the catalog id is not given in configuration the id from the default catalog will be automatically used.

The same function is also available for the store ID. In case a store ID is not given it can also be retrieved from the *IBM WCS* but a given config value for `store.id` takes precedence.

Content Settings

Each site can have one single shop configuration (see the Blueprint site concept). That means only shop items from exactly that shop instance (with a particular view to the product catalog) can be interwoven to the content elements of that site.

At least the `config.id` must be configured for the site root page (see the *Local Settings* tab) within a struct property named `livecontext.store.config`. This `config.id` maps to a named store configuration mentioned above (configured via Spring). The Spring configuration itself provides all other connection relevant values.

Table 5.7. *config.id*

Name	Type	Description	Example	Required
config.id	String Property	The configuration ID defined in Spring configuration	myStore	true

All other store configuration settings, like the `store.id` will be taken from the Spring configuration. But it is also supported to overwrite such settings within the content settings.

The concrete store related IDs (`store.id` and `catalog.id`) can also be dynamically retrieved from the *IBM WCS*. As long as a `store.name` and `catalog.name` value is available in the configuration (Spring or content settings) the corresponding IDs will be retrieved dynamically.



Redefine the Currency

A popular example would be the usage of a base configuration in Spring referenced by the `config.id` but with the variation of the locale and currency for each site (default currency of `myStore` is `USD`).

Name	Type	Description	Example	Required
<code>config.id</code>	String Property	The configuration ID defined in Spring configuration	<code>myStore</code>	<code>true</code>
<code>currency</code>	String Property	The currency for all product prices	<code>EUR</code>	<code>false</code>

Table 5.8. Currency configuration

Be aware, that the locale is also part of each shop context. It is defined by the locale of the site. That means all localized product texts and descriptions have the same language as the site in which they are included and one specific currency.



Enabling Dynamic Pricing

Dynamic price rendering is disabled by default. If this feature is not used on *IBM WCS* side then it is not necessary to turn it on on *CMS* side. It avoids an additional call to *IBM WCS* that is not needed in such a scenario.

But if you use personalized price rules in *IBM WCS* then it is necessary to switch this feature on. For price rules on contract bases (where the prices are the same for all members of the group) you do not necessarily need to enable this feature.

Name	Type	Description	Example	Required
<code>config.id</code>	String Property	The configuration ID defined in Spring configuration	<code>myStore</code>	<code>true</code>

Table 5.9. Currency configuration

Name	Type	Description	Example	Required
dynamicPricing.enabled	Boolean Property	Personalized product prices enabled	true	false

Please see [Section 5.4, “Connecting with an IBM WCS Shop” \[207\]](#) to get the information how the dynamic prices can be switched on on *IBM WCS* side.

Tenant specific Configuration

Per default only one *IBM WebSphere Commerce Server Management Center* system can be configured per *Content Application Engine*. If you want to connect to different *IBM WebSphere Commerce Server Management Center* hosts per site for example (dev, staging), you need to duplicate all URL configuration properties the hostname is part of via Spring. Since this would multiply the amount of your configuration properties, LiveContext 2 provides a mechanism to replace placeholder tokens within your configured URLs etc. with values defined in the current *StoreContext* at runtime.

For example within the `component-lc-ecommerce-ibm.properties` the following is defined:

```
livecontext.ibm.wcs.url=http://${livecontext.ibm.wcs.host}
livecontext.ibm.wcs.rest.path=/wcs/resources
livecontext.ibm.wcs.rest.url=${livecontext.ibm.wcs.url}${livecontext.ibm.wcs.rest.path}
```

Instead of the global host configuration you want to connect different *IBM WebSphere Commerce Server Management Center* environments per site:

```
livecontext.ibm.wcs.url=http://{livecontext.ibm.wcs.host}
livecontext.ibm.wcs.rest.path=/wcs/resources
livecontext.ibm.wcs.rest.url=${livecontext.ibm.wcs.url}${livecontext.ibm.wcs.rest.path}
```

You need to add the following to your site specific store configuration:

```
<util:map id="auroraStoreConfigDev">
  ...
  <entry key="livecontext.ibm.wcs.host" value="myhost"/>
</util:map>
```

After doing that the URL token will be replaced with the values of you current store configuration at runtime: `livecontext.ibm.wcs.rest.url` >> `http://myhost/wcs/resources`

The configuration keys must exactly match with the token defined in your URL configuration. If you add custom properties make sure to use the replacement mechanism `CommercePropertyHelper`:

```
public String getCustomWcsUrl() {  
    return CommercePropertyHelper.replaceTokens(customWcsUrl,  
StoreContextHelper.getCurrentContext());  
}
```

5.5 Link Building for Fragments

If you include CoreMedia fragments into WCS pages, these fragments might contain links to WCS pages; a link to an Augmented Category, for example. Depending on the scenario that you use, this link should lead to a page rendered by the CAE (content-led scenario) or to a page rendered by the WCS (commerce-led scenario). The latter is named "deep link".

Overview

A use case for deep links might be the following: You have an existing commerce solution with carefully styled category and product pages. While you want to switch to *CoreMedia DXP 8* in order to enhance your site with editorial content, there is no need to port the commerce pages to *CoreMedia DXP 8*. Instead, you want to reuse the existing pages (possibly enhanced with *CoreMedia DXP 8* fragments).

Configuring deep links

CoreMedia DXP 8 supports two settings to switch to deep links for categories and products:

Properties for deep link activation

→ `livecontext.policy.commerce-product-links`

→ `livecontext.policy.commerce-category-links`

The settings are at the root channel of each site. The default setting is `true`, which means that the CAE creates deep links to the product or category pages of the IBM WCS. However, for links to other content types, such as HTML, CSS or JavaScript, links to the CAE will be generated. Also, URLs to dynamic resources (`UriConstants.Prefixes.PREFIX_DYNAMIC`) won't be converted to JSON. See [Section 5.6, "Enabling Preview of Commerce Category Pages in Studio" \[215\]](#) to learn how to enable the preview for WCS pages in *Studio*.

Default setting "true"

The settings are evaluated by the `LiveContextPageHandlerBase` and its sub-classes.

If a setting is `true`, the corresponding `@Link` method creates links to IBM WCS, so there is no need for a matching `@RequestMapping` method. If it is `false`, the `@Link` method creates CAE links. So you must keep the according `@RequestMapping` method in sync with changes to the URL pattern and provide (or customize) the `ProductPageHandler` or `ExternalNavigationHandler` classes. See also the Section 4.3, "The CAE Web Application" in *CoreMedia Content Application Developer Manual* for request handling and link building.

Link building and request handling

Format of Deep Links

Each `lc:include` requests an HTML fragment via HTTP from the CAE. Every link of a fragment that is requested by the WCS from the CAE is processed by `LiveContextLinkTransformer` classes. The transformer is only applied for fragment

How deep links are build

requests. Depending on the document type the link should be generated for, an absolute CAE URL is generated or a JSON string is returned. Each of these JSON objects contains at least the values of the constants `LiveContextLinkResolver.OBJECTTYPE` and `LiveContextLinkResolver.RENDERTYPE` and the ID of the content.

For example, the HTML fragment contains a link to a `CMArticle` document. Instead of rendering the regular link, for example

```
http://localhost/blueprint/servlet/page/perfectchef/magazine-spring/spring-salads-1888
```

the corresponding Link generated by the `LiveContextLinkResolver` would look like:

```
a href="<!--CM {
  "id":"cm-1696-1888",
  "renderType":"url",
  "externalSeoSegment":"spring-salads-1888",
  "objectType":"content"}
CM-->" ...
```

The `CoreMedia Fragment Connector` will parse the JSON, identify the object type and rendering type and apply a template to render a commerce link that points to the parameterized Struts action `CoreMediaContentURL`. For the given example, the template `Content.url.jsp` will be used, applied by the pattern "`<OBJECT_TYPE>.<RENDER_TYPE>.jsp`". The JSP file will render a commerce URL afterwards:

```
http://localhost/webapp/wcs/stores/servlet/CoreMediaContentURL?
storeId=10202&externalSeoSegment=spring-salads-1888&
urlRequestType=Base&langId=-1&catalogId=10051
```

The SEO feature has not been configured for this example, otherwise the `externalSeoSegment` value would be used to render a SEO friendly URL.




Other templates are located in the folder `workspace\Stores\WebContent\Widgets-CoreMedia\com.coremedia.commerce.store.widgets.CoreMediaContentWidget\impl\templates` by default. The path is configurable via property `com.coremedia.widget.templates` in `coremedia-connector.properties`. New templates can be added by extending the `CMObjectLiveContextLinkResolver` in the *Blueprint* workspace. Custom object types can be added, depending on the document type of the content or its property values. Also, additional rendering types can be defined for an object type. Using this templating mechanism, it is possible to support different layouts for content depending on its context.

5.6 Enabling Preview of Commerce Category Pages in Studio

When you have links in your content that point to an `Augmented Category` content item, *CoreMedia DXP 8* allows you to build links instead, that link to the corresponding commerce category page. The feature depends on the configuration flag `livecontext.policy.commerce-category-links` which is located in the `LiveContext` settings document, that is linked to the root channel of a site. If this Boolean property is set to `true`, the CAE will render links for augmented categories that point to the corresponding category page of the commerce system instead of rendering a regular CAE page link (see [Section 5.5, “Link Building for Fragments” \[213\]](#)). In this case, the Studio user wants to see the commerce category page in the preview too.

In order to enable the preview of Commerce category pages in Studio, proceed as follows:

1. Open the `CommonJSToInclude.jspf` file and ensure that `${jsAssetsDir}javascript/CoreMedia/coremedia-pbe.js` is included if `_cm_page_pbe_pageData` is not empty.
2. Open the `application.properties` file of the `studio-webapp`. The `studio.previewUrlWhitelist` property must contain the commerce URL (including the port, for example `*coremedia.com` or `http://localhost:40080`). Be aware that this property overwrites the `studio.previewUrlPrefix` property, so you have to add the default CAE preview URL to the `studio.previewUrlWhitelist` property too.

 *Configure in the CoreMedia system*

If your IBM WCS shop storefront uses any clickjacking prevention features (for example, `X-Frame-Options` (see http://www-01.ibm.com/support/knowledge-center/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/tasks/tseiframerestrictx-frame.htm?lang=en for details), please make sure to allow the shop preview (IBM WCS Staging-/Authoringserver) being embedded as an `iframe` within *CoreMedia Studio*.



5.7 Enabling Contract Based Preview

In *Studio* you can preview the effect of different WCS contracts on your pages. To enable the preview you have to do the setup in the WCS system and the CoreMedia system.

Setup within the IBM WCS

The *IBM WCS Feature Enhancement Pack 8* enables the management of B2B extended sites, clients and organizations. In order to enable contract based preview in *CoreMedia Digital Experience Platform 8* you need to create a dedicated commerce user in your *IBM WCS*. The user credentials (username and password) will be used in a public way and send as plaintext in an URL call. Furthermore, the user should be authorised to use the contract you intend to preview within *Studio*.

For more information on how to configure commerce users and organizations please refer to the *IBM WCS* documentation.

You have to enable the cookie leveler from within your `WCDE-INSTALL/work space/Preview/WebContent/WEB-INF/web.xml` file of your preview web application, which is described in [Section 3.4.5, “Configuring the Cookie Domain” \[65\]](#)

Setup within Blueprint

For contract based preview of shop pages in *Studio*, you can configure contracts to the test personas of a B2B enabled site.

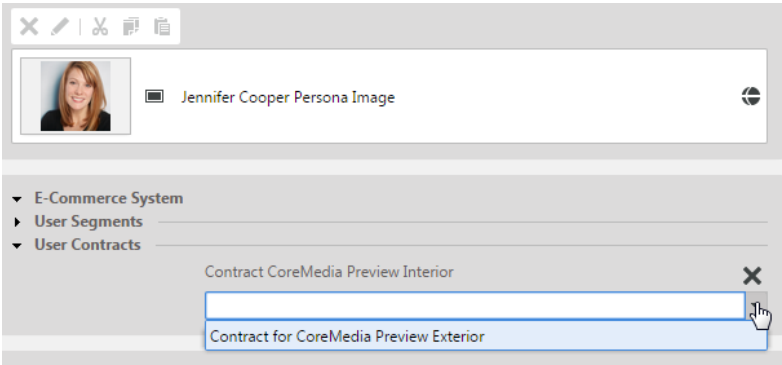


Figure 5.16. Edit Commerce Contracts in Test Persona

If you edit an Augmented Page in *Studio* and select a test persona with a configured contract, the preview will automatically login as a dedicated service user for contract preview and redirects to the current shop page with the selected contract. The following screenshots show the same Augmented Page with no test persona selection compared to contract based preview.

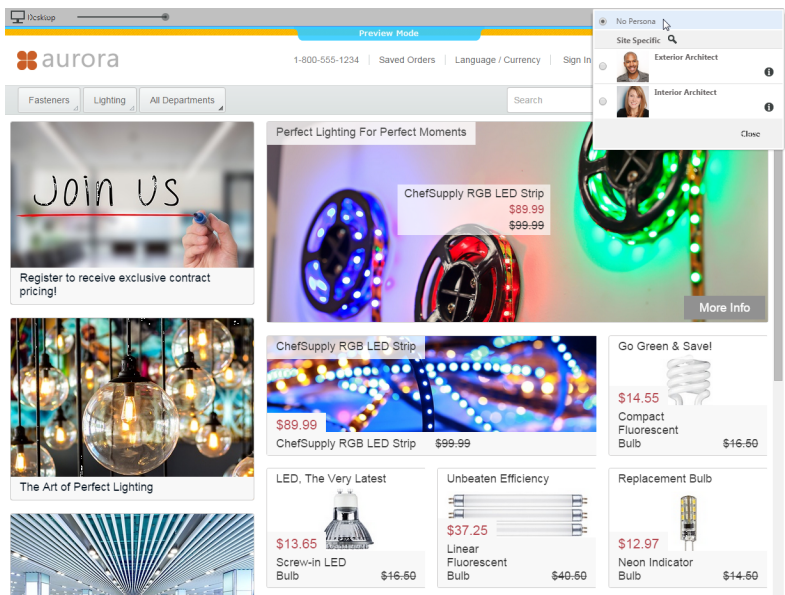


Figure 5.17. Preview Augmented Page no Test Persona

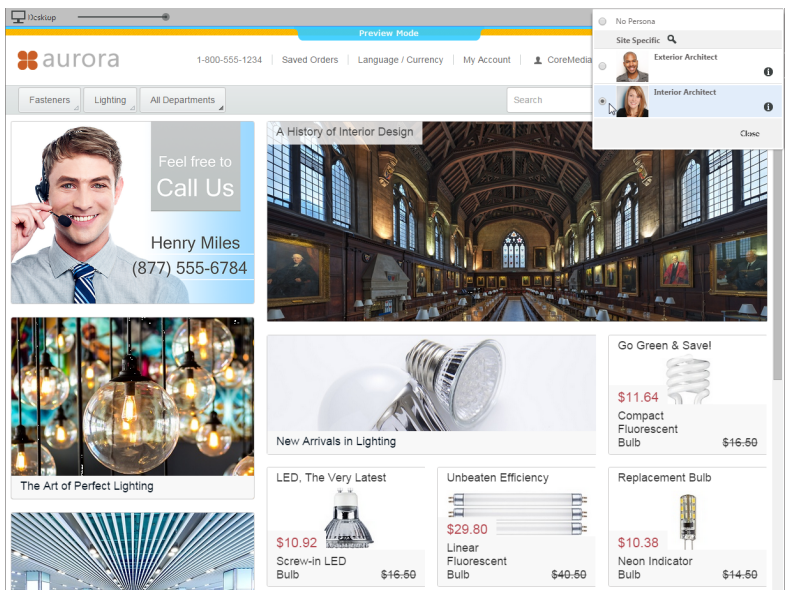


Figure 5.18. Preview Augmented Page with Contracts in Test persona

These properties are important for B2B contract based personalization and can be configured in `application.properties`:

<code>livecontext.ibm.contract.preview.credentials.username</code>	
Description	The service user used for contract based shop preview in b2b scenarios
Example	preview
<code>livecontext.ibm.contract.preview.credentials.password</code>	
Description	Password of the contract preview user
Example	changeme

Table 5.10. Properties for B2B contract based personalization

5.8 The e-Commerce API

The *e-Commerce API* is a Java API provided by *CoreMedia DXP 8* that can be used to build shop applications. Various services allow you to access the e-Commerce system for different tasks:

- The `CatalogService` can be used to access the product catalog in many ways: traverse the category tree, products by category, various product and category searches.
- With the `PriceService` you can access prices: list prices and dynamic offer prices.
- The `AvailabilityService` lets you access the inventory of the e-Commerce system to check the availability of Stock Keeping Units (SKUs).
- The `MarketingSpotService` gives you access to *IBM WCS* e-Marketing Spots, a common method to use marketing content (product teasers, images, texts) depending on the customer segments.
- The `SegmentService` lets you access customer segments, for example, the customer segments the current user is a member of.
- Use the `CartService` to manage orders.
- The `WorkspaceService` lets you retrieve the list of existing workspaces. A workspace is a concept to prepare changes in a separated branch/environment. A selected workspace is part of the current store context you work with.

The *e-Commerce API* is used internally to render catalog-specific information into standard templates. Furthermore, the Studio Library integration makes use of the API to browse and work with catalog items. If you develop your own shop application you will use the API in your templates and/or business logic (handlers and beans).

The following key points will give you a short overview of the components that are also involved. They build up an infrastructure to bootstrap a connection to a commerce system and/or perform other supportive tasks.

- The `Commerce` class is the essential part of the bootstrap mechanism to access a commerce system. You can use it to create a connection to your commerce system.
- The `CommerceConnectionInitializer` is used to initialize a request specific commerce connection. The resolved connection is stored in a thread local variable.
- The `CommerceBeanFactory` creates `CommerceBeans` whose implementation is defined via Spring. It is also used by the services to respond service

calls, for example, instances of `Product` and/or `Category` beans. You can integrate your own commerce bean implementations via Spring (inheriting from the original bean implementation and place your own code would be a typical pattern).

- The `StoreContextProvider` can retrieve an applicable `StoreContext` (the shop configuration that comprises information like the shop name, the shop id, the locale and the currency).
- The `UserContextProvider` is responsible to retrieve the current `UserContext`. Some operations, like requesting dynamic price information, demand a user login. These requests can be made on behalf of the requesting user. User name and user id are then part of the user context.
- The `CommerceIdProvider` is able to format and parse references to commerce items. References to commerce items will be possibly stored in content, like a product teaser stores a link to the commerce product.

Commerce beans are cached on time basis. Cache time and capacity can be configured via Spring.

Please refer to the Javadoc of the `Commerce` class as a good starting point on 'How to use the *e-Commerce API*'.

5.9 Commerce Cache Configuration

The CoreMedia system uses caching to provide a faster access to various e-Commerce entities (that is, products, categories, etc.). These entities will automatically be cached when used by the CoreMedia system. The unified API cache keys are used for caching the commerce entities.

The *e-Commerce API* defines cache classes for each entity. These are used to define default capacity and cache time. Each of the default values can be adapted to the needs of your system environment by overwriting the corresponding properties. The following overview lists all default values available for configuration. Each value can be overwritten in the corresponding *CoreMedia Blueprint* application.properties file.

Please note that the CoreMedia system also performs an active event based cache invalidation (see also [Section 3.4.13, “Event-based Commerce Cache Invalidation” \[78\]](#)).

```
# e-Commerce Cache parameter for each Commerce related CacheClass
# cache time will be set in seconds, capacity in number of instances
#
# Note, for each language and currency variant a separate cache
# instance is used!

# Product and SKU instances
livecontext.ecommerce.cache.product.time=3600
livecontext.ecommerce.cache.product.capacity=10000

# Category instances
livecontext.ecommerce.cache.category.time=3600
livecontext.ecommerce.cache.category.capacity=10000

# number of lists containing top categories
livecontext.ecommerce.cache.topCategoryLists.time=3600
livecontext.ecommerce.cache.topCategoryLists.capacity=100

# number of lists containing sub categories
livecontext.ecommerce.cache.subCategoryLists.time=1800
livecontext.ecommerce.cache.subCategoryLists.capacity=1000

# number of product lists for categories (products as direct
# members of a category)
livecontext.ecommerce.cache.productListsByCategory.time=3600
livecontext.ecommerce.cache.productListsByCategory.capacity=500

# Marketing Spot instances
livecontext.ecommerce.cache.marketingSpot.time=3600
livecontext.ecommerce.cache.marketingSpot.capacity=5000

# number of lists containing all marketing spots
livecontext.ecommerce.cache.marketingSpotLists.time=3600
livecontext.ecommerce.cache.marketingSpotLists.capacity=100

# dynamic/personalized price instances
livecontext.ecommerce.cache.dynamicPrice.time=300
livecontext.ecommerce.cache.dynamicPrice.capacity=10000

# static/list price instances
livecontext.ecommerce.cache.staticPrice.time=300
```

```

livecontext.ecommerce.cache.staticPrice.capacity=10000

# availability infos for all products
livecontext.ecommerce.cache.availability.time=300
livecontext.ecommerce.cache.availability.capacity=10000

# user segment instances
livecontext.ecommerce.cache.segment.time=3600
livecontext.ecommerce.cache.segment.capacity=5000

# number of lists containing all user segments
livecontext.ecommerce.cache.segmentLists.time=3600
livecontext.ecommerce.cache.segmentLists.capacity=100

# number of segment lists for all users
livecontext.ecommerce.cache.segmentsByUser.time=60
livecontext.ecommerce.cache.segmentsByUser.capacity=1000

# contract instances
livecontext.ecommerce.cache.contract.time=3600
livecontext.ecommerce.cache.contract.capacity=500

# number of contract lists for all users
livecontext.ecommerce.cache.contractsByUser.time=60
livecontext.ecommerce.cache.contractsByUser.capacity=200

# number of lists containing all workspaces
livecontext.ecommerce.cache.workspaceLists.time=3600
livecontext.ecommerce.cache.workspaceLists.capacity=100

# number of previewTokens used be all editors
# rule of thumb for capacity: 5 x number of editors
livecontext.ecommerce.cache.previewToken.time=3600
livecontext.ecommerce.cache.previewToken.capacity=1000

# user is logged in info that is used for request
# rule of thumb for capacity: number of expected concurrent requests
livecontext.ecommerce.cache.userIsLoggedIn.time=10
livecontext.ecommerce.cache.userIsLoggedIn.capacity=500

# commerce user instances (caching for request purpose only)
livecontext.ecommerce.cache.commerceUser.time=10
livecontext.ecommerce.cache.commerceUser.capacity=500

# store info instances
livecontext.ecommerce.cache.storeInfo.time=3600
livecontext.ecommerce.cache.storeInfo.capacity=100

# contract ids for all users
livecontext.ecommerce.cache.contractIdsByUser.time=3600
livecontext.ecommerce.cache.contractIdsByUser.capacity=1000

```

5.10 Studio Integration of the IBM WebSphere Commerce Content

CoreMedia Digital Experience Platform 8 offers an integration of *IBM WebSphere Commerce Server* systems. Each content site can be configured with a specific shop instance to deliver content pages mixed with e-Commerce catalog items. The term "e-Commerce catalog items" means all items that live only in the e-Commerce catalog. Nevertheless, these elements are to be interwoven with content on mixed pages.

From classical shop pages, like a product catalog ordered by categories or product detail pages up to landing pages or homepages, all grades of mixing content with catalog items are conceivable. The approach followed in this chapter, assumes that items from the catalog will be linked or embedded without having stored these items in the CMS system. Catalog items will be linked typically and not imported (importless integration).

- [Section 5.10.1, "Catalog View in CoreMedia Studio Library" \[223\]](#) gives a short overview over the Catalog mode in the *Studio* Library.
- [Section 5.10.2, "WCS Management Center Integration in CoreMedia Studio" \[227\]](#) gives a short overview over the WCS Management Center integration in *CoreMedia Studio*.
- [Section 5.10.3, "WCS Preview Support Features" \[227\]](#) gives a short overview over the IBM WCS preview functions that are supported in *CoreMedia Studio*.
- [Section 5.10.4, "Working with WCS Workspaces" \[230\]](#) shows how *CoreMedia Studio* supports the IBM WCS Workspaces.
- [Section 5.10.5, "Augmenting WCS Content" \[231\]](#) describes how you augment WCS content in the commerce-led scenario in *CoreMedia Studio*.

5.10.1 Catalog View in CoreMedia Studio Library

When the connection to the *IBM WCS* system and a concrete shop for a content site are configured as described in [Section 5.4, "Connecting with an IBM WCS Shop" \[207\]](#) the *Studio* Library shows the e-Commerce catalog to browse product categories, products and marketing spots in the e-Commerce catalog and to search for products, product variants and marketing spots. After the editor has selected a preferred site with a valid store configuration the catalog view will be enabled and the catalog will be shown in the Library:

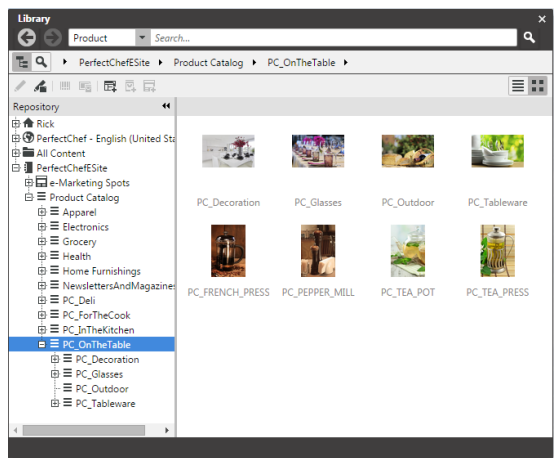


Figure 5.19. Library with catalog in the tree view

These catalog items can be accessed and assigned to various places within your content. For example, an *e-Commerce Product Teaser* document can link to a product or product variant from the catalog. The product link field (in *e-Commerce Product Teaser* documents) can be filled by drag and drop from the library in catalog mode.

Linking a content (like the *e-Commerce Product Teaser*) to a catalog item leads to a link that is stored in the CMS document and references the external element. Apart from the external reference (in the case of the *IBM WCS* it is typically a persistent identifier like the part number for products) no further data will be imported (importless integration).

While browsing through the catalog tree you can also open a preview of a category or a product from the library. It can be achieved by a double-click on a product in the product list or by activating the context menu on a product or a category (right click on the desired item) and choosing the entry "Open in Tab" from the context menu as shown in the pictures below.

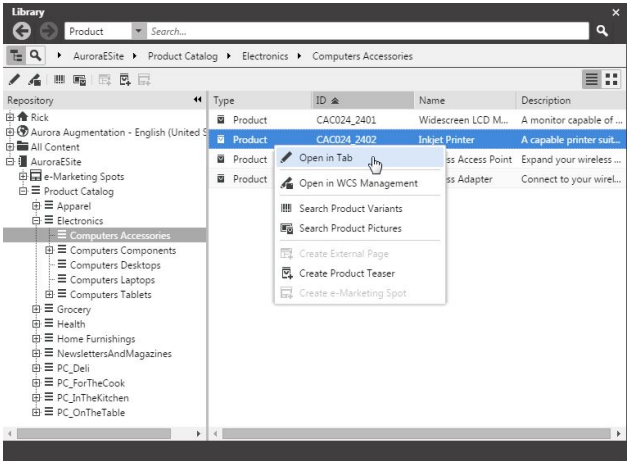


Figure 5.20. Open Product in tab

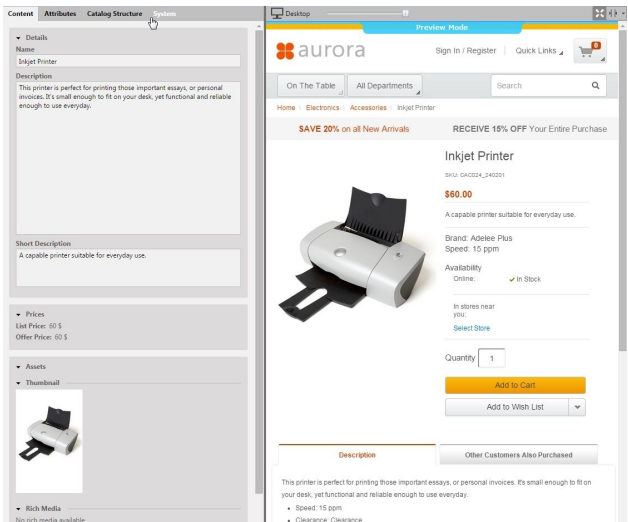


Figure 5.21. Product in tab preview

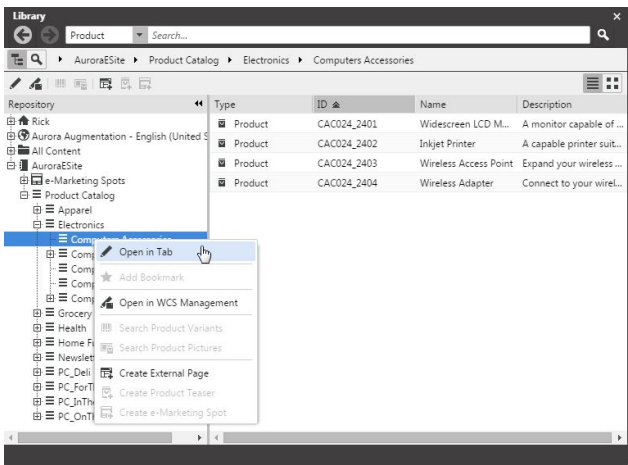


Figure 5.22. Open Cat-
egory in tab

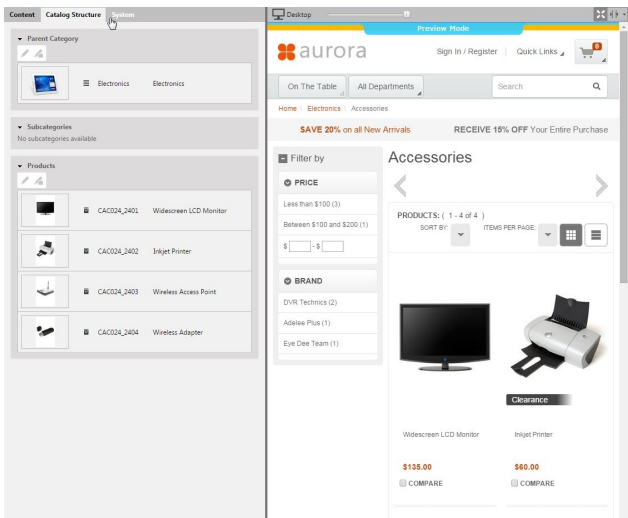


Figure 5.23. Category
in tab preview

In addition to the ability to browse through the e-Commerce catalog in an explorer-like view it is also possible to search for products, variants and marketing spots from catalog. As for the content search if you are in the catalog mode and you type a search keyword into the search field and press **Enter**, the search in the e-Commerce system will be triggered and a search result displayed.

5.10.2 WCS Management Center Integration in CoreMedia Studio

In addition to the e-Commerce catalog library integration you can directly access the *IBM WebSphere Commerce Server Management Center* from *CoreMedia Studio*. A context menu action on a product, product variant, category or e-marketing spot opens the item in a window within *CoreMedia Studio* where catalog item properties can be edited directly. This applies to all components in *CoreMedia Studio* which represent a product, product variant, category or e-marketing spot. Categories in the library do not open in Management Center by double click as this is the default behavior for navigation in the library tree.

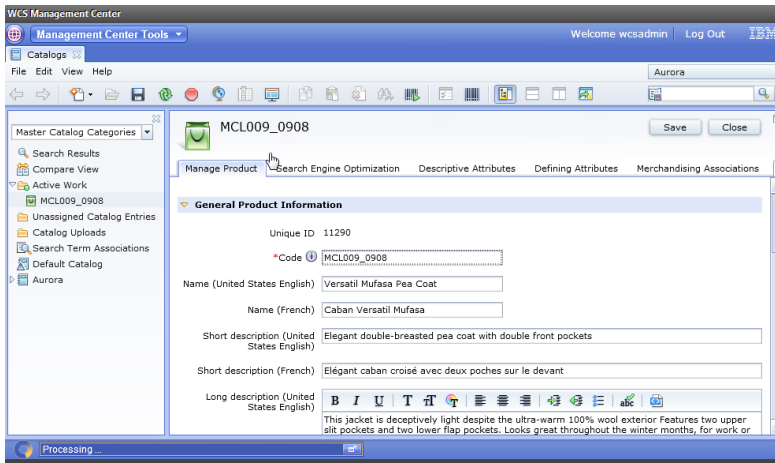


Figure 5.24. Management Center in Studio

Known restriction:

- ➔ Up to FEP 7, the only supported web browsers are Internet Explorer and Firefox as these are supported web browsers for *IBM WebSphere Commerce Server Tools*. Since FEP 8 Chrome is also supported.
- ➔ Currently there is no Single Sign On implemented between *CoreMedia Studio* and *Management Center*. You have to login to the *Management Center* with your *IBM WCS* login credentials.



5.10.3 WCS Preview Support Features

CoreMedia Studio supports a variety of *IBM WCS* preview functions directly:

➔ Time based preview (time travel)

When a preview date is set in *CoreMedia Studio*, it sets the virtual render time to a time in the future. If the currently previewed page contains content from *IBM WCS*, it is desirable that also these content reflects the given preview time. That could be a marketing spot containing activities with different validity time ranges. A specific activity could be valid only after a certain time or a marketing teaser that announces a happy hour could be another example.

If such data is requested from *IBM WCS* within the context of a *CoreMedia* page, the preview date is also sent to *IBM WCS* as a genuine *IBM WCS* preview token. The *IBM WCS* recognizes the transmitted preview date and renders a control on top of the page that lets you inspect the currently active settings. [Figure 5.25, “Time based preview affects also the IBM WCS preview” \[228\]](#) gives an example.

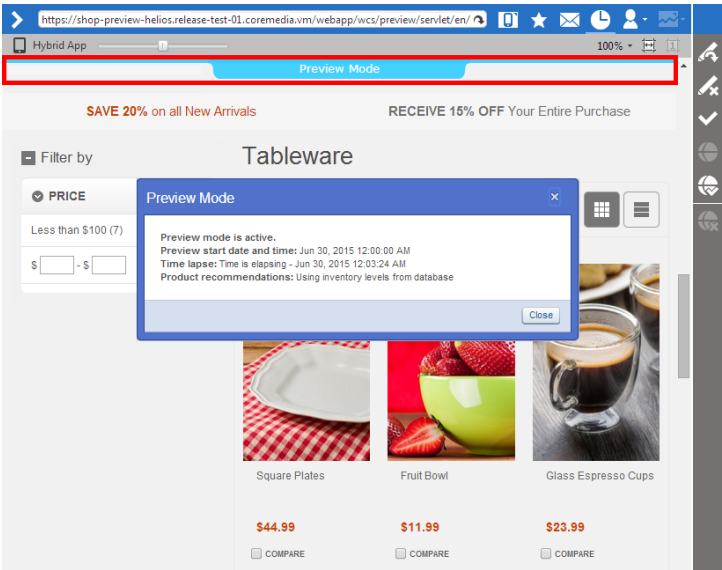


Figure 5.25. Time based preview affects also the IBM WCS preview

➔ Customer segment based preview

The commerce segment personalization is not available in IBM WCS (FEP6).

FEP7+

Another case where editors need preview support is the creation of personalized content. That is, content is shown depending on the membership in specific customer segments. In addition to the existing rules, you can define rules that are based on the belonging to customer segments that are main-

tained by IBM WCS. These commerce segments will be automatically integrated and appear in the chooser if you create a new rule in a personalized content. For a preview, editors can use test personas which are associated with specific customer segments.

Figure 5.26, “Test Persona with Commerce Customer Segments” [229] shows an example where the test persona is female and has been already registered.

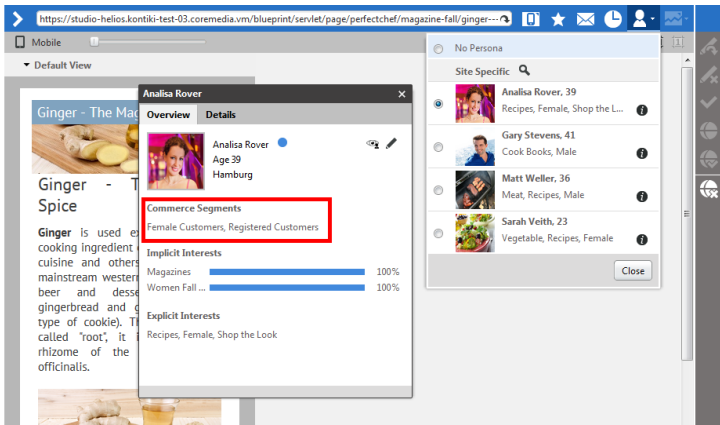


Figure 5.26. Test Persona with Commerce Customer Segments

Such preview settings apply as long as they are not reset by the editor.

The test persona document can be created and edited in CoreMedia Studio. The customer segments available for selection will be automatically read from the IBM WebSphere Commerce Server.

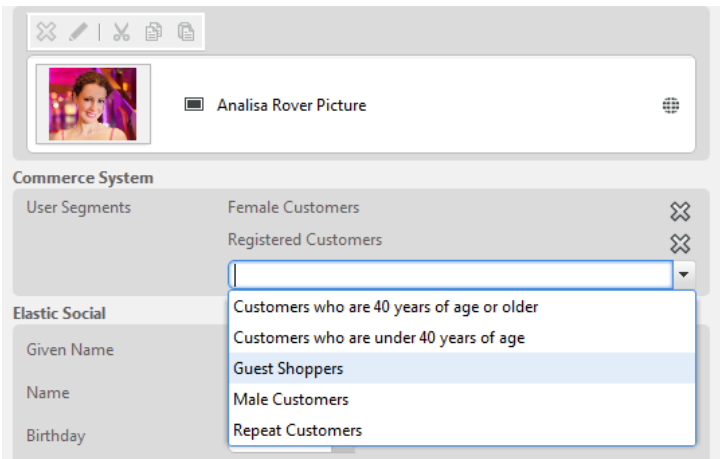


Figure 5.27. Edit Commerce Segments in Test Persona

Personalized content based on commerce customer segmentation rules can be used in both, the content-led scenario and the commerce-led scenario. If the *CoreMedia CAE* is rendering *IBM WCS* content, like catalog items or marketing spots, the given user ID is also sent to the *IBM WCS*. So all content that is received from the *IBM WCS* is delivered within the context of the current *IBM WCS* user.

The *IBM WCS* segments that the current user belongs to are available during the rendering process within a *CoreMedia CAE*. Thus, content from the *CoreMedia* system can also be filtered based on the current *IBM WCS* segments.

In the other direction, if the personalized content is integrated within a content fragment on a *IBM WCS* page, the current *IBM WCS* user is also transmitted as a parameter. Thus, the *CoreMedia* system can retrieve the connected customer segments from the *IBM WCS* in order to perform commerce segment personalization within the supplied content fragments.

→ B2B Contract based preview

CoreMedia Adaptive Personalization has been extended to support a personalized site preview for B2B contracts from *IBM WCS*. A two-step configuration needs to be applied in order to use the B2B contract based preview within *Studio*. See [Section 5.7, “Enabling Contract Based Preview” \[216\]](#) to learn how to enable contract based preview.

5.10.4 Working with WCS Workspaces

CoreMedia Studio supports working with *IBM WCS Workspaces*. If the *Workspaces* feature is enabled in *IBM WCS* and if you work on a workspace to prepare changes in a separated space (that are invisible to other users) the same workspace can be chosen in *CoreMedia Studio*.

You can select the workspace in the *User Preferences Dialog*. The setting is available only if *Workspaces* are enabled and at least one workspace exists in the *IBM WCS* system.

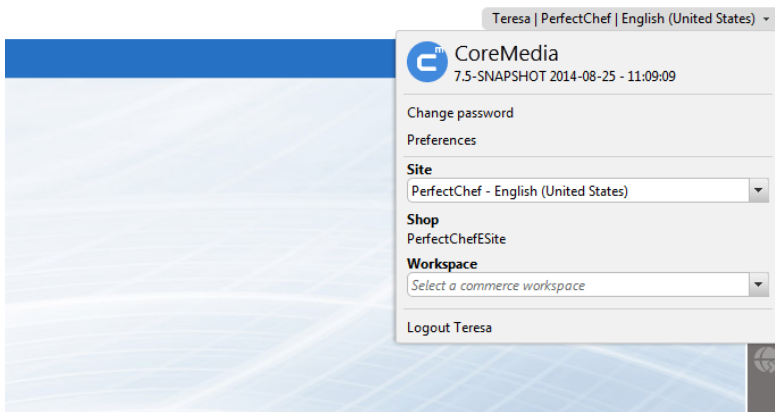


Figure 5.28. Workspaces selector in User Preferences Dialog

The selection of an IBM WCS workspace in *CoreMedia Studio* lets you access shop items that may only exist in a workspace. On the CMS side there is no mechanism that separates the edited content elements accordingly. If you change the selected workspace in *CoreMedia Studio* or if you reset it by selecting *No workspace* all editorial changes still remain. That means on CMS side only one global space is used and no separated workspace specific Projects. That can lead to situations where possibly not working references are left in CMS content (references to catalog items that are not visible for other users). There is no common procedure to deal with that. You should be aware of the issue and address it through organizational precautions, like editing in separated content areas up to work with separate content sites.



5.10.5 Augmenting WCS Content

In the commerce-led scenario you can augment pages from the WCS, such as PDPs or categories, with content from the CMS system. The following sections describe the steps required in *Studio*.

In general, extending a shop page with CMS content comprises the following steps, which will be explained in the corresponding sections. It is supposed that the WCS and CMS systems are connected as describe in [Section 5.4, “Connecting with an IBM WCS Shop” \[207\]](#).

1. Augment the root channel of the WCS catalog as described in [Section “Augmenting the Root Node” \[232\]](#)
2. In the CMS create a document of type `Augmented Category` or `Augmented Page`.

3. When you augment a category, the connection between the category and the `Augmented Category` content is automatically created. For the `Augmented Page` you have to create this connection manually via a property, which contains a part of the URL to the page.
4. In the `Augmented Category` or `Augmented Page` choose a page layout that corresponds to the shop page layout. It should contain all the placements for which there are `CoreMedia Content Widgets` defined on the WCS side.
5. Drop the augmenting content into the right placements of the `Augmented Category` or `Augmented Page` content item. That is, into a placement whose name corresponds with the name defined in the `CoreMedia Content Widget`.

Augmenting the Root Node

The root channel of your site is a content of type `Augmented Page`. This page will hold all elements that are used to augment the home page of the WCS. The root channel also serves as the fallback location for fragments for which no more specific pages are found. Therefore, most of the settings of a site are directly linked to the root channel.

Root channel of the site

If the shop connection is properly configured, you will see an additional top level entry in the *Studio* library that is named after your store (*AuroraESite*, for instance). Below this node you can open the *Product Catalog* with categories and products. The *Product Catalog* node also represents the root category of a catalog.

Catalog view in Studio

To have a common ancestor for all augmented catalog pages, the root node of the WCS catalog must be augmented. You can augment the root category by clicking *Augment Category* in the context menu of the root category. An augmented category content opens up, where you can start to define the default elements of your catalog pages, like the page layouts for the *Category Overview Pages (COP)* and *Product Detail Pages (PDP)* and first content elements. All sub categories, augmented or not, will inherit these settings. See Section 6.2.3, “Adding CMS Content to Your Shop Pages” in *CoreMedia Studio User Manual* for more information.

Augmented catalog root

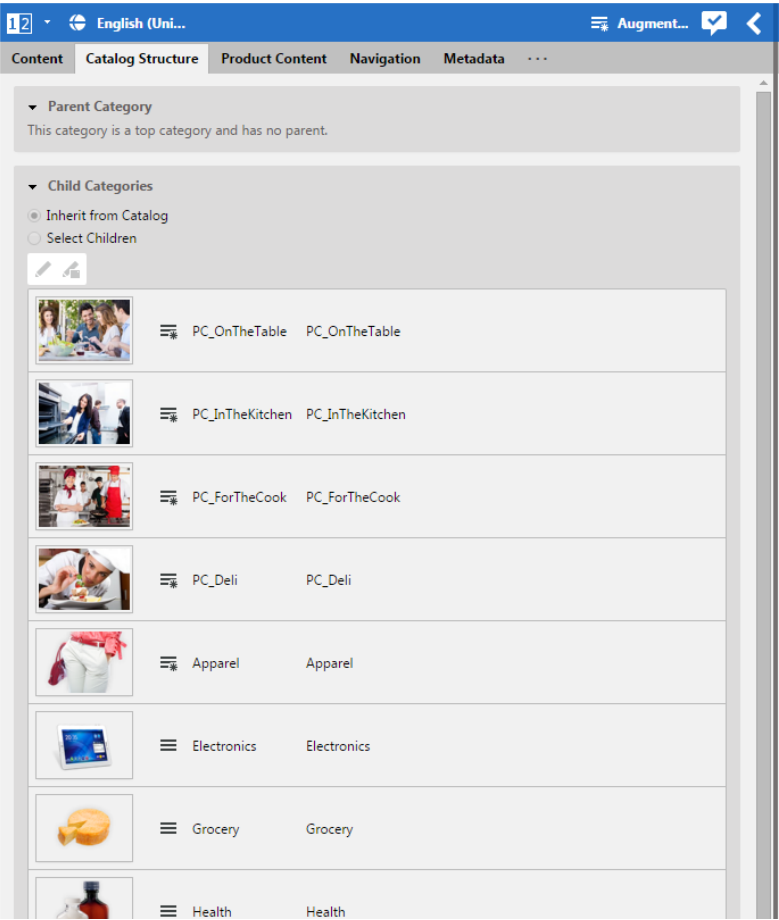


Figure 5.29. Catalog structure in the catalog root content item

Now, you can start augmenting sub categories or Product Detail Pages of the catalog. All content and settings are inherited down in this hierarchy. However, you cannot add non-augmented pages to the navigation and inheritance hierarchy. This is different in the content-led scenario.

Selecting a Layout for an Augmented Page

CoreMedia Digital Experience Platform 8 comes with a predefined set of page layouts. Typically, this selection will be adapted to your needs in a project. By selecting a layout an editor specifies which placements the new page will have, which of them can be edited and how the placements are arranged generally. It should correspond to the actual shop page layout. All usable placements should be addressed. The

placement names must match the placement names used in the slot definition on the shop side.

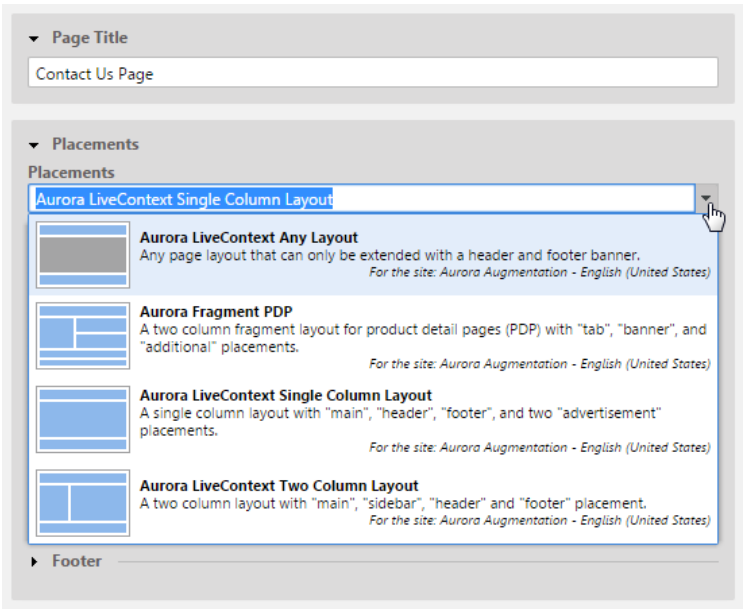


Figure 5.30. Choosing a page layout for a shop page

If you augment a category, the corresponding `Augmented Category` document contains two page layouts: the one in the `Content` tab is applied to the Category Overview Page and the other in the `Product Content` tab is used for all Product Detail Pages. Both layouts are taken from the root category. The layouts that are set there form the default layouts for a site. Hence, they should be the most commonly used layouts. If you want something different, you can choose another layout from the list.

Finding CMS Content for Category Overview Pages

A category overview page is a kind of landing page for a product category. If a user clicks on a category without specifying a certain product, then a page will be rendered that introduces a whole product category with its subcategories. Category overview pages contain typically a mix of promotional content like product teasers, marketing content (that can also be product teasers but of better quality) or other editorial content. You can use the `CoreMedia Content Widget` in the commerce-led scenario in order to add content from the CoreMedia CMS to the category overview page.

Category overview pages

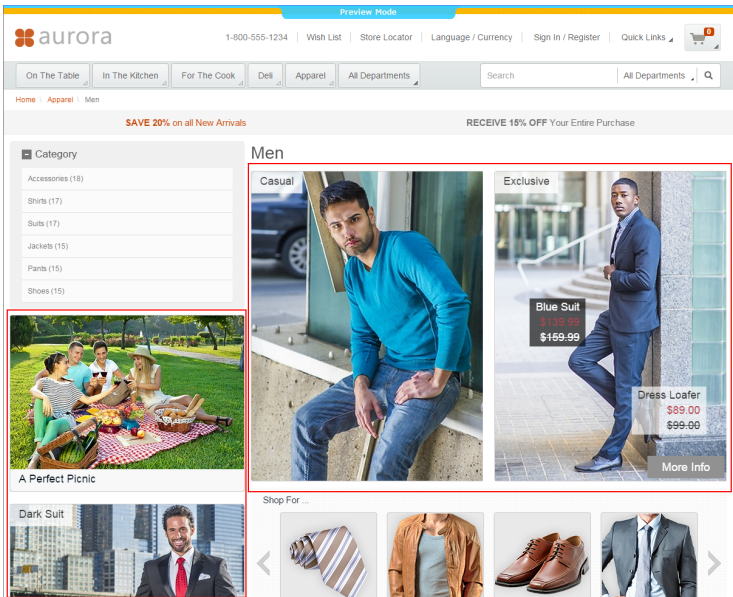


Figure 5.31. Category overview page with CMS content

When a category page contains the *CoreMedia Content Widget*, then, on request, the current category ID and the name of the placement configured in the *Content Widget* are passed to the CoreMedia system. The CoreMedia system uses this information to locate the content in the CoreMedia repository that should be shown on the category overview page.

Information passed to the CoreMedia system

CoreMedia DXP 8 tries to find the required content with a hierarchical lookup using the category ID and placement name information. The lookup involves the following steps:

Locating the content in the CoreMedia system

1. Select the *Augmented Page* that is connected with the IBM store (see [Section 5.4, "Connecting with an IBM WCS Shop" \[207\]](#) on how to connect an IBM shop with *CoreMedia DXP 8*).
2. Search in the catalog hierarchy for an *Augmented Category* content item that references the catalog category page that should be augmented and that contains a placement with the name defined in the *CoreMedia Content Widget*.
 - a. If there is no *Augmented Category* for the category, search the category hierarchy upwards until you find an *Augmented Category* that references one of the parent categories.
 - b. If there is no *Augmented Category* at all, take the site root *Augmented Page*.
3. From the found *Augmented Category* or *Augmented Page* take the content from the placement which matches the placement name defined in the *Content Widget*.

Figure 5.32, “Decision diagram” [236] shows the complete decision tree for the determination of the content for the category overview page or the product detail page (see below for the product detail page).

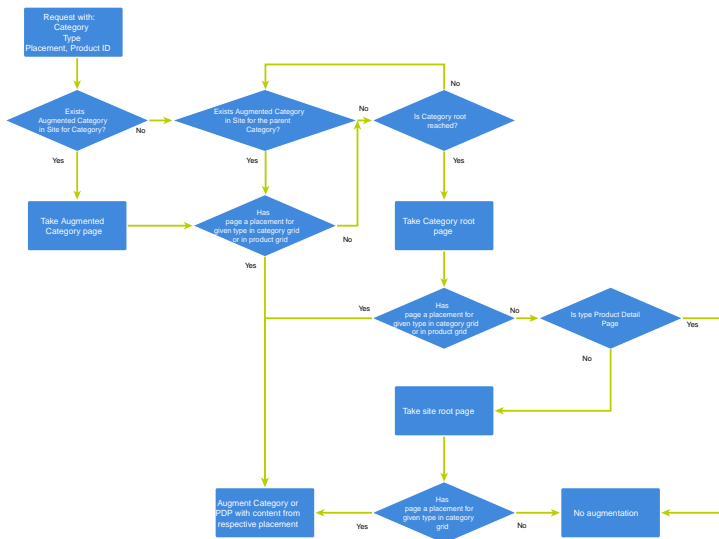


Figure 5.32. Decision diagram

Keep the following rules in mind when you define content for category overview pages:

- ➔ You do not have to create an *Augmented Category* for each category. It's enough to create such a page for a parent category. It is also quite common to create pages only for the top level categories especially when all pages have the same structure.
- ➔ You can even use the site root's *Augmented Page* to define a placement that is inherited by all categories of the site.
- ➔ If you want to use a completely different layout on a distinct page (a landing page's layout, for example, differs typically from other page's layouts), you should use different placement names for the "Landing Page Layout", for example with a `landing-page` prefix (as part of the technical identifier in the struct of the layout document). This way, pages below the intermediate landing page, which use the default layout again, can still inherit the elements from pages above the intermediate page (from the root category, for instance), because the elements are not concealed by the intermediate page.

Finding CMS Content for Product Detail Pages

Product detail pages give you detailed information concerning a specific product. That includes price, technical details and many more. You can enhance these pages with content from the CoreMedia system by adding the *CoreMedia Content Widget* similar to the category overview page.

Product detail pages

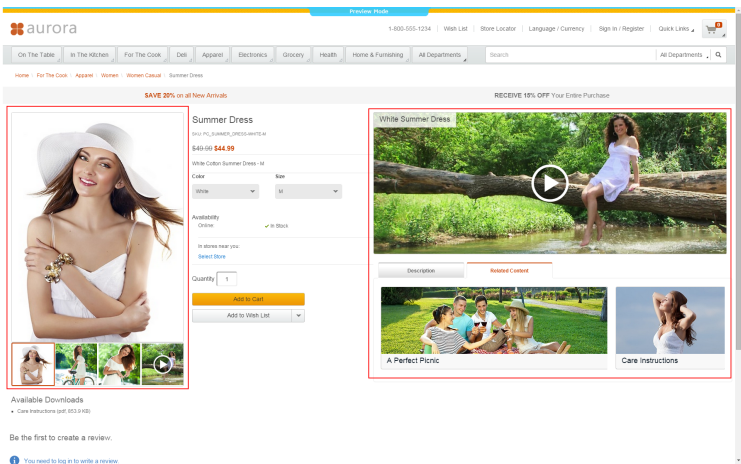


Figure 5.33. Product detail page with CMS content highlighted by the red border

Similar to the category overview pages, the Category ID and placement name are passed to *CoreMedia DXP 8* in order to locate the content.

Information passed to the CoreMedia system.

For product detail pages, *CoreMedia DXP 8* uses the same lookup as described for the category overview page. That is, an *Augmented Category* content item is searched by category ID that matches the category of the product. There is only one difference; the site root *Augmented Page* content item is not considered as a default for the product detail page.

Locating the content in the CoreMedia system

The content to augment is taken from a separate page grid of the *Augmented Category*, called *Product Content*.

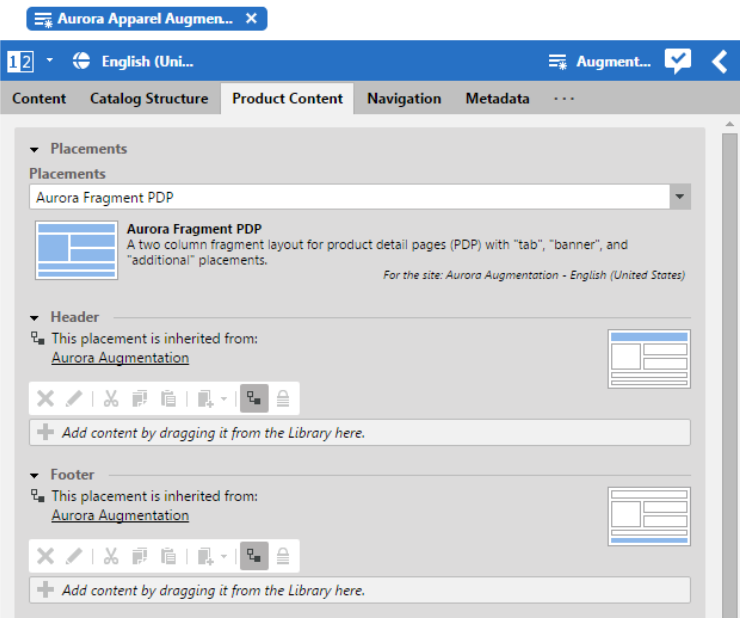


Figure 5.34. Page grid for PDPs

Adding CMS Assets to Product Detail Pages

You can enhance product detail pages with assets from the CoreMedia system by adding the *CoreMedia Asset Widget*.

Product detail pages

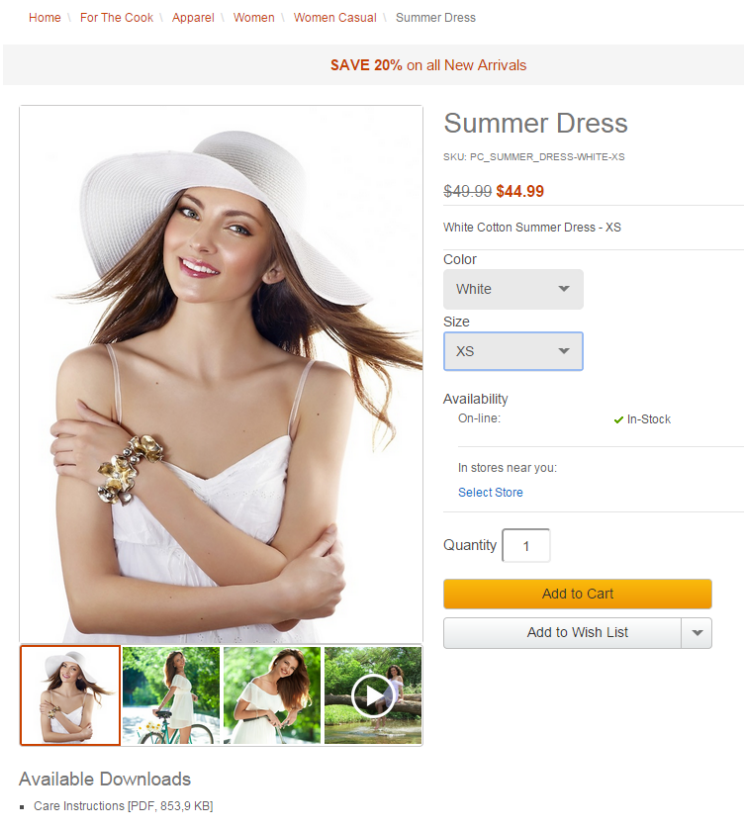


Figure 5.35. Product detail page with CMS assets

The Product ID and orientation are passed to *CoreMedia DXP 8* in order to locate and layout the assets.

Information passed to the CoreMedia system.

To find assets for product detail pages, *CoreMedia DXP 8* searches for the picture content items which are assigned to the given product. These items are then sorted in alphabetical order. See [Section 8.7, "Advanced Asset Management" \[443\]](#) for details.

Locating the assets in the CoreMedia system

Adding CMS Content to non-catalog Pages (Other Pages)

Non-catalog pages (or "Other Pages") like "Contact Us", "Log On" or even the homepage are shop pages, which can also be extended with CMS content. The "homepage" case is quite obvious. The need to enrich the homepage with a custom layout and a mix of promotional and editorial content is very clear. However, the less prominent pages can also profit from extending with CMS content. For example,

Non Catalog Pages (Other Pages)

context-sensitive hotline teasers, banners or personalized promotions could be displayed on those pages.

You can augment a non-catalog page with *Studio* using the preview's context menu. In the *Studio* preview, navigate to the non-catalog page that should be augmented, right-click its page title and select **Augment page** from the context menu.

You can also perform the following steps using the common content creation dialog:

1. Make sure, that the layout of the page in the WCS contains the *CoreMedia Content Widget*.
2. Create a document of type *Augmented Page* and add it to the *Navigation Children* property of the site root content.
3. Enter the ID (from the URL) of the other page into the *External Navigation* field of the *Augmented Page*.
4. Optional: Set the *External URI Path* if special URL building is needed.

In the following example a banner picture was added to an existing "Contact Us" shop page. To do so, you have to create an *Augmented Page*, select an corresponding page layout and put a picture to the *Header* placement.

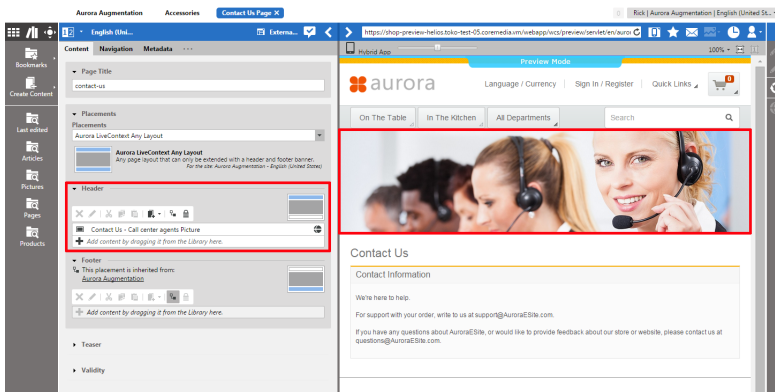


Figure 5.36. Example: Contact Us Pagegrid

The case to augment a non-catalog page with *CoreMedia Studio* differs only slightly from augmenting a catalog page. You use *Augmented Page* instead of *Augmented Category* and instead of linking to a category content, you have to enter a page ID in the *External Navigation* field. The page ID identifies the page unambiguously. Typically it is the last part of the shop URL-path without any parameters.

```
https://<shop-host>/en/aurora/contact-us
```

Difference between the augmentation of catalog and other pages

The URL above would have the page id `contact-us` that will be inserted into the *External Navigation* on the *Navigation* tab. In case of a standard "SEO" URL without the need of any parameters the *External URI Path* field can be left empty.

Figure 5.37. Example: Navigation Settings for a simple SEO Page

When the URL to a WCS page is not a standard SEO URL but contains, for example, additional parameters, you can add this additional information via the *External URI Path* field (see [Figure 5.38, "Example: Navigation Settings for a custom non SEO Form" \[242\]](#)). This is necessary in order to get the *Studio* preview for the augmented page or for links rendered from the CMS. Therefore, if you have entered the correct URL, you will see the page in the preview.

URLs of non-SEO pages

In the *External URI Path* field, you redefine the URL path starting from `/en/aurora/...` and add required parameters. For example the advanced search page does not use the standard SEO path and in turn it has additional parameters:

```
.../AdvancedSearchDisplay?catalogId=10152&langId=-1&storeId=10301
```

Some of the standard parameters are well known and can be replaced by tokens, because they are very typical for all such URLs. In order to flexibly copy these URLs to other sites with different shop configurations the following tokens can be used:

Table 5.11. config.id

Token	Description
storeId	The current store id.
catalogId	The current catalog id.
langId	The current language id.

Tokens have to be enclosed with curly braces. In case of the Advanced Search Page it would be possible to enter to following String into the *External URI Path*:

```
/AdvancedSearchDisplay?catalogId={catalogId}&langId={langId}&storeId={storeId}
```

▼ Navigation

Navigation Children

✕ ✎ ✂ 📄 📁 🖨

+ Add content by dragging it from the Library here.

Visibility

☐ Hidden in Navigation and Sitemap

☐ Hidden in Sitemap

▼ Enhanced Page

Page Type

☐ Catalog Page (for Category or Product)

☒ Other Page

External Navigation

AdvancedSearchDisplay

External URI Path

/AdvancedSearchDisplay?catalogId={catalogId}&langId={langId}&storeId={storeId}

Figure 5.38. Example: Navigation Settings for a custom non SEO Form

Be aware that the property *External Navigation* must be unique within all other "Other Pages" of that site. Otherwise the rendering logic is not able to resolve the matching page correctly. A validator in *CoreMedia Studio* displays an error message, if a collision of duplicate *External Navigation* values occurs. Your navigation hierarchy can differ from the "real" shop hierarchy. There is also no need to gather all pages below the root page. You can completely use your custom hierarchy with additional pages in between, that are set *Hidden in Navigation* but can be used to define default content for are group pages.



Special Case: Homepage

The home page of the site is the main entry point, when you want to augment an *IBM WCS* catalog. In the commerce-led scenario, it is a content item of type *Augmented Page*. While in the content-led scenario it is a *Page*.

Special Case:
Homepage

The *External Navigation* field can be left empty. The homepage is anyway the last instance that will be chosen if no other page can be found to serve a fragment request.

The *External URI Path* field is also likely to remain empty, unless the shop site is to be accessible with an URL, which still has a path component (e.g. `../en/aurora/home.html`). But in most cases you wouldn't want that.

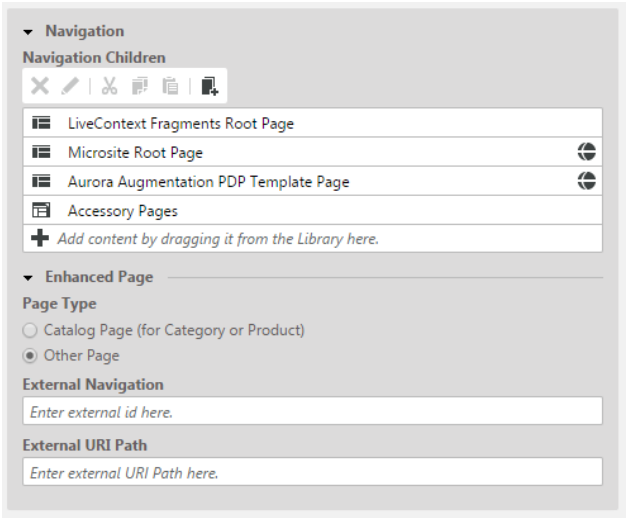


Figure 5.39. Special Case: Navigation Settings for the Homepage

6. CoreMedia DXP 8 e-Commerce Blueprint - Functionality for Websites

This chapter describes the *CoreMedia DXP 8* content type structure and the resulting website structure.

- [Section 6.2, “Basic Content Management” \[248\]](#) describes aspects of the content type model of *CoreMedia Blueprint*.
- [Section 6.3, “Website Management” \[264\]](#) describes all features relevant for website management, such as layout, search and navigation.
- [Section 6.4, “Website Development with Themes” \[311\]](#) describes how you will work with content when you develop your website.
- [Section 6.5, “Localized Content Management” \[332\]](#) describes all aspects of multi-site management.
- [Section 6.6, “Workflow Management” \[356\]](#) describes all aspects of multi-site management.

6.1 Overview of e-Commerce Blueprint

The *e-Commerce Blueprint* provides a modern, appealing, highly visual website template that can be used to start a customization project. It demonstrates the capability to build localizable, multi-national, experience-driven e-Commerce web sites. Integration with IBM WebSphere Commerce ships out of the box. Other e-Commerce systems can be integrated via the CoreMedia e-Commerce API as a project solution.

Two integration patterns are available with the product:

- e-Commerce-led fragment-based approach shown in the Aurora B2C and B2B store examples
- Experience-led hybrid blended approach shown in the Perfect Chef store example

Based on a fully responsive, mobile-first design paradigm, the *e-Commerce Blueprint* leverages the Masonry dynamic grid framework and the Freemarker templating framework. It scales from mobile via tablet to desktop viewport sizes and uses the CoreMedia Adaptive and Responsive Image Framework to dynamically deliver the right image sizes in the right aspect ratios and crops for each viewport.

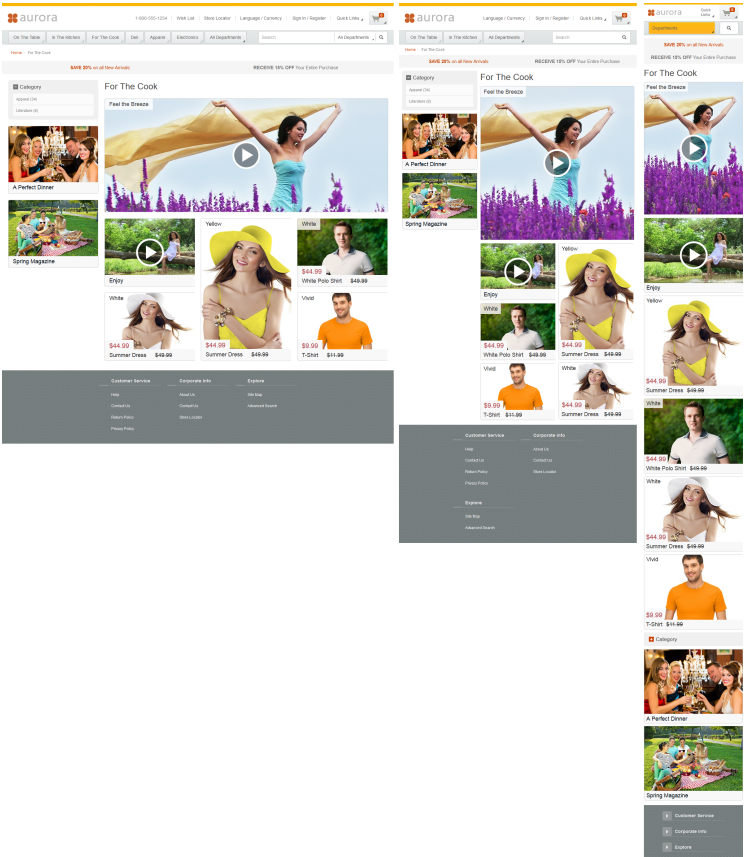


Figure 6.1. Aurora category page for different devices: desktop, tablet, mobile

The responsive navigation can blend e-Commerce as well as content categories and content pages seamlessly and in any user-defined order that does not have to follow the catalog structure. Navigation nodes with URLs to external sites can be added in the content.

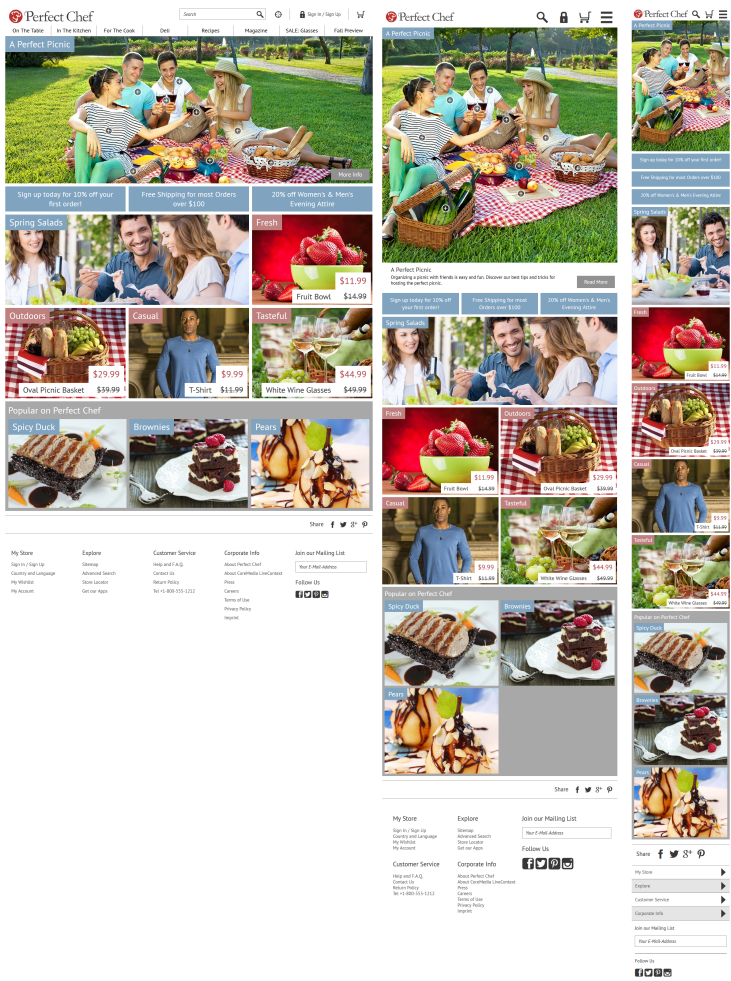


Figure 6.2. Perfect Chef homepage for different devices: desktop, tablet, mobile

6.2 Basic Content Management

The basis of the information structure of a CoreMedia system are content types. Content types organize your content and form a hierarchy with inheritance. See the [Content Server Manual], [Developing a Content Type Model](#) and [Section 9.6, "Content Type Model" \[476\]](#) for more details.

CoreMedia Blueprint comes with a comprehensive content type model that covers the following topics:

- Common content such as Articles or Pictures.
- Placeholder types that you can use to link to e-Commerce content
- Taxonomies are used to tag content.

6.2.1 Common Content Types

Requirements

An appealing website does not only contain text content but has also images, videos, audio files or allows you to download other assets such as brochures or software.

In addition, current websites aim to reuse content in different contexts. An article about the Hamburg Cyclastics might appear in Sports, Hamburg and News section, for example. An image of the St. Michaelis church (the "Hamburger Michel") on the other hand might appear in Articles about sights in Hamburg or religion. Nevertheless, it's not a good idea to copy the article to each section or the image to each article because this is error prone, inefficient and wastes storage.

Therefore, content should be reusable across different contexts (different sites, customer touchpoints for instance) by just applying the context specific layout and without having to duplicate any content. This increases the productivity by reducing redundancy and keeps management effort at a minimum.

Solution

CoreMedia Blueprint is shipped with content types that model common digital assets such as articles, images, videos or downloads. All these types inherit from a common parent type and can be used interchangeably. In addition, none of these types has fixed information about its context so that it can be used repeatedly and everywhere in your site. The context is first determined through the page which links to the document or through the position in the folder hierarchy of the website (see [Section 6.3.2, "Navigation and Contexts" \[265\]](#) for more details).

Common Content Types

CoreMedia Blueprint defines the following types for common content. Using CoreMedia’s object oriented content model projects can define their own content types or add to the existing ones.

Table 6.1. Overview of Content Types for common content

CMArticle	
UI-Name	Article
Description	Contains mostly the textual content of a website combined with images.
CMPicture	
UI-Name	Picture
Description	Stores images of the website. The editor can define different crops of the image which can be used in different locations of the website.
CMVideo	
UI-Name	Video
Description	Stores videos which can be viewed on the website.
CMAudio	
UI-Name	Audio
Description	Stores audio/podcast information which can be heard on the website.
CMDownload	
UI-Name	Download
Description	Stores binary data for download. You can add a description, image and the like.
CMGallery	
UI-Name	Gallery
Description	Aggregates images via a linklist. You can add a description, teaser text and the like.

e-Commerce Placeholder Types

Blueprint comes with some additional content types required to build representations of entities of an e-Commerce system.

Table 6.2. e-Commerce Content Types

CMProductTeaser	
UI-Name	Product Teaser

Description	A teaser for products of the e-Commerce system. It inherits from <code>CMTeasable</code>
CMMarketingSpot	
UI-Name	e-Marketing Spot
Description	A placeholder for an e-Marketing spot. It inherits from <code>CMTeasable</code> .
CMExternalChannel	
UI-Name	Category Placeholder
Description	Documents of this type are used to build a CMS representation of commerce categories. It inherits from <code>CMAbstractCategory</code> which in turn inherits from <code>CMChannel</code> .
CMExternalPage	
UI-Name	Placeholder for other WCS pages such as Help pages or the main page.
Description	Documents of this type are used to build a CMS representation of other commerce pages. It inherits from <code>CMChannel</code> .

e-Commerce Content Properties

A short description of the properties provided for e-Commerce scenarios is provided below.

Table 6.3. Overview e-Commerce Content Properties

externalId	
UI-Name	External ID
Description	The ID of the corresponding entity in the e-Commerce system. For a <code>CMProductTeaser</code> this id is the technical id of the product in the catalog.
localSettings.shopNow	
UI-Name	'Shop Now' flag
Description	This Boolean flag is stored in the local settings of the document types <code>CMProductTeaser</code> and <code>CMExternalChannel</code> and is used in the content-led scenario. If enabled the 'Shop Now' overlay is visible for product teasers. This configuration is extendable via <code>CMExternalChannels</code> and may be overwritten for every <code>CMProductTeaser</code> .

Common Content Properties

All common content types extend the abstract type `CMTeasable` to share common properties and functionality. Teasable means that you can show for each content

that inherits from `CMTeasable` a short version that "teases" the reader to watch the complete article, site or whatever else.

A short description of the core properties of content is provided below. Properties specific for certain Blueprint features such as teaser management etc. are described in their respective sections (follow the link in the *Description* column).

Table 6.4. Overview Common Content Properties

title	
UI-Name	(Asset) Title
Description	The name or headline of an asset, for example the name of a download object or the headline of an article.
detailText	
UI-Name	Detail Text
Description	A detailed description, for example the article's text, a description for a video or download.
teaserTitle, teaserText	
UI-Name	Teaser Title and Text
Description	The title and text used in the teaser view of an asset. See Section 6.3.9, "Teaser Management" [286] .
pictures	
UI-Name	Pictures
Description	A reference to <code>CMPicture</code> items that illustrate content. Examples include a photo belonging to the article, a set of images from a video etc. Usage of the pictures depends on the rendering. In <i>Blueprint</i> the pictures are used for teasers and detail views of content.
related	
UI-Name	Related Content
Description	The related content list refers to all items that an editor deems related to the content. For an article for a current event this list could include a video describing of the event, a download with event brochure, an audio/podcast file with an interview with the organizers, an image gallery with photos of the previous event and many more.
keywords	
UI-Name	Keywords
Description	Keywords for this content. <i>CoreMedia Blueprint</i> currently uses keywords as meta information for the HTML <code><head></code> .
subjectTaxonomy	
UI-Name	locationTaxonomy

Description	Tags for this content. See Section 6.2.3, “Tagging and Taxonomies” [254] for details.
viewType	
UI-Name	Layout Variant
Description	The layout variant influences the visual appearance of the content on the site. It contains a symbolic reference to a view that should be used when the content is rendered. For more information see Section 6.3.7, “View Types” [282]
segment	
UI-Name	URL Segment
Description	A descriptive segment of a URL for this content. Used for SEO on pages displaying the content. See Section 6.3.15, “URLs” [300]
locale, master, masterVersion	
UI-Name	Locale, Master, Master Version
Description	See Section 6.5, “Localized Content Management” [332] for details. Properties for the Localization of this asset.
validFrom, validTo	
UI-Name	Valid From, Valid To
Description	Meta information about the validity time range of this content. Content which validity range is not between validFrom and validTo will not be displayed on the website. See Section 6.3.17, “Content Visibility” [301] for details.
notSearchable	
UI-Name	Not Searchable Flag
Description	Content with this flag will not be found in end user website search. See Section 6.3.21, “Website Search” [308] for details.

Media Content

The abstract content type `CMMedia` defines common properties for all media types. Media types for content such as pictures (`CMPicture`), video (`CMVideo`), audio (`CMAudio`), and HTML snippets (`CMHTML`) inherit from `CMMedia`.

data	
UI-Name	Data

Table 6.5. *CMMedia* Properties

Description	The core data of the content. Either a <code>com.coremedia.cap.common.Blob</code> or in the case of CMHTML a <code>com.coremedia.xml.Markup</code> .
copyright	
UI-Name	Copyright
Description	Allows you to store arbitrary copyright information in a string property.
alt	
UI-Name	Alternative Representation
Description	Allows managing alternative representations of an image, for example a description of an image that can be used to enable a website accessible for the visually impaired.
caption	
UI-Name	Caption
Description	The caption of a content. Unused property in <i>Blueprint</i> .

A common feature of all `CMMedia` objects is the ability to generate and cache transformed variants of the underlying object (see `CMMedia#getTransformedData`). This ability is extensively used for rendering images without the need to store image variants and renditions as distinct blobs in the system.

6.2.2 Adaptive Personalization Content Types

Adaptive Personalization extends *Blueprint* with the following content types:

→ **Personalized Content** (`CMSelectionRules`)

Personalized Content enables an editor to explicitly determine under which conditions a certain Content is shown. Conditions can be combined with **AND** and **OR** operators to create complex expressions. At runtime, theses Conditions are evaluated against the provided contexts.

→ **Personalized Search** (`CMF13NSearch`)

Personalized Search documents can be used to augment search engine queries with context data. The result is a dynamic list of Content.

→ **User Segments** (`CMSegment`)

User Segments let an editor predefine sets of conditions to be (re-)used in Personalized Content, thereby grouping your website's visitors. For example one can imagine a User Segment called "Teenage Early Birds". This could then aggregate the conditions

- "(logged in) user is older than 14"
- "(logged in) user is younger than 20"
- "It is earlier than 10 am"
- Test User Profiles (`CMUserProfile`)

Test User Profiles are artificial contexts under the control of the editors. They can be used to test the CAE's rendering when creating Personalized Content. Typically, Test User Profiles are used to simulate certain website visitors containing the corresponding context properties.

6.2.3 Tagging and Taxonomies

Requirements

Most websites define business rules that require content to be classified into certain categories. Typical examples include use cases such as "Display the latest articles that have been labeled as press releases" or "Promote content tagged with 'Travel' and 'London' to visitors of pages tagged with 'Olympic Games 2012'" etc.

Keywords or tags are common means to categorize content. Employing a controlled vocabulary of tags can be more efficient than allowing free-form keyword input as it helps to prevent ambiguity when tagging content. Furthermore, a system that supports the convenient management of tags in groups or hierarchies is required for full editorial control of the tags used within a site.

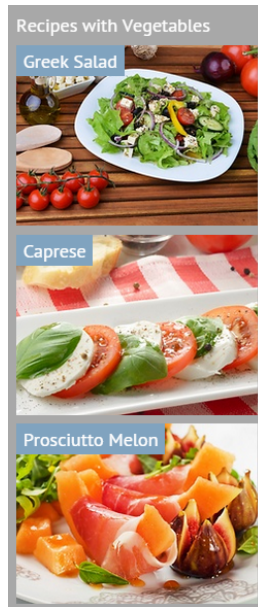


Figure 6.3. Dynamic list of articles tagged with "Vegetables"

Solution

Blueprint currently uses tag information in various ways:

- It is possible to use the taxonomies of a content item as conditions for dynamic lists of content (such as "5 latest articles tagged with 'London'").
- In CoreMedia Adaptive Personalization tags can be used to gather information about the topics a site visitor is interested in (see `TaxonomyInterceptor`).
- In CoreMedia Adaptive Personalization tag information representing the interests of visitors can be used to define user segments, conditions for personalized selection rules and personalized searches.
- It is possible to display related content for a content item based on content that shares a similar set of tags (see `CMTeasableImpl#getRelatedBySimilarTaxonomies`).

In *CoreMedia Blueprint* tags are represented as `CMTaxonomy` content items which represent a controlled vocabulary that is organized in a tree structure. *CoreMedia Blueprint* defines two controlled vocabularies: Subject and location taxonomies that can be associated with all types inheriting `CMLinkable`.

Taxonomy Management

Subject taxonomies can be used to tag content with "flat" information about the content's topic (such as Olympic Games 2012). They can also enrich assets with hierarchical categorization for fine-grained drill down navigation (such as Hardware / Printers / Laser Printer). Subject Taxonomies are represented by the content type `CMTaxonomy` which defines the following properties:

value	
Type	String
Description	Name of this taxonomy node
children	
Type	Link list
Description	References to subnodes of this taxonomy node
externalReference	
Type	String
Description	Reference of an equivalent entity in an external system in the form of an ID / URI etc.

Table 6.6. *CMTaxonomy Properties*

Location taxonomies allow content to be associated with one or more locations. Location taxonomy hierarchies can be used to retrieve content for a larger area even if it is only tagged with a specific element within this area ("All articles for 'USA'" would include articles that are tagged with the taxonomy node North America / USA / Louisiana / New Orleans). Location taxonomies are represented by the content type `CMLocTaxonomy` which inherits from `CMTaxonomy` and adds geographic information for more convenient editing and visualization of a location.

latitudeLongitude	
Type	String
Description	Latitude and longitude of this location separated by comma
postcode	
Type	String
Description	The post code of this location

Table 6.7. *Additional CMLocTaxonomy Properties*

The taxonomy administration editor can be used to create a taxonomy and build a tree of keywords.

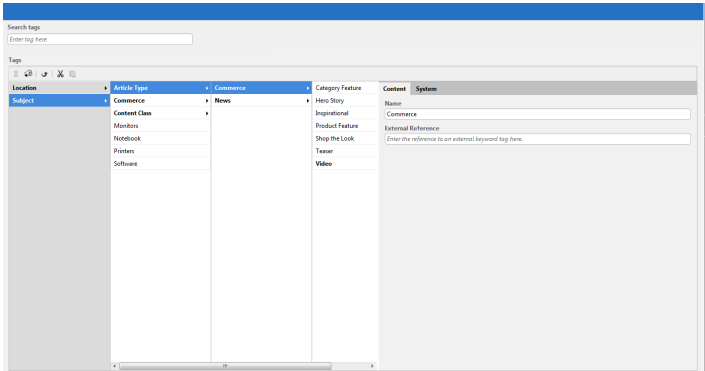


Figure 6.4. Taxonomy Administration Editor

The taxonomy administration editor displays taxonomy trees and provides drag and drop support and the creation and deletion of keywords.

Taxonomy Assignment

To enable tagging of content two properties are available the `CMLinkable` content type.

subjectTaxonomy	
Type	Link list
Description	Subject(s) / topic(s) of that content item
locationTaxonomy	
Type	Link list
Description	Geographic location(s) of that content item

Table 6.8. CMLinkable Properties for Tagging

Property	Type	Purpose
subjectTaxonomy	Link list	Subject(s) / topic(s) of that content item
locationTaxonomy	Link list	Geographic location(s) of that content item

Table 6.9. CMLinkable Properties for Tagging

Editors can assign taxonomies to content items using *CoreMedia Studio* and the Blueprint taxonomy property editor. It allows for the following:

- ➔ adding/removing references to taxonomy

- ➔ autocompletion
- ➔ suggestions

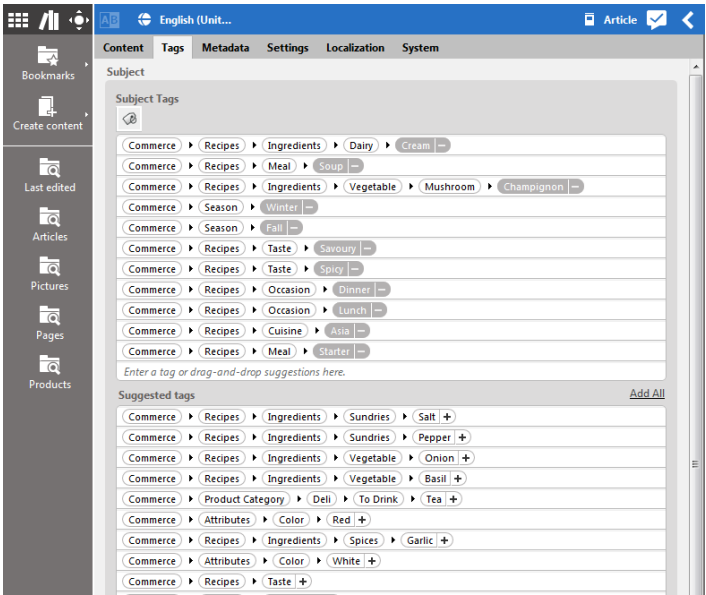


Figure 6.5. Taxonomy Property Editor

The user can add taxonomy keywords to the corresponding property link list using the taxonomy property editor. The editor also provides suggestions that are provided by the *OpenCalais* integration or a simple name matching algorithm. The strategy type can be configured in the preferences dialog of *CoreMedia Studio*.

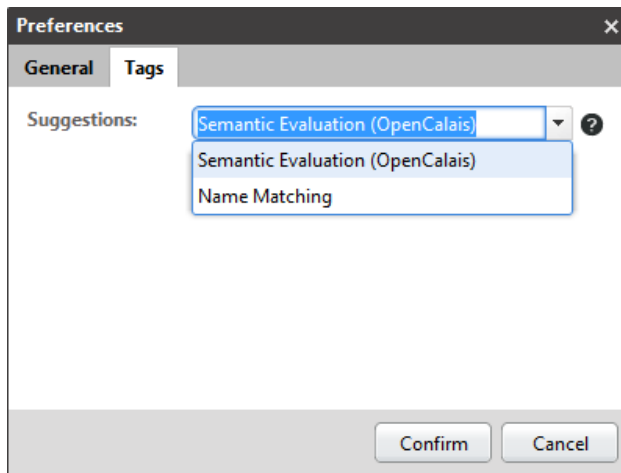


Figure 6.6. Taxonomy Studio Settings

How taxonomies are loaded

A Blueprint taxonomy tree is built through content items located in a specific folder of the content repository. As a default strategy for building a taxonomy tree, the taxonomy REST service of the taxonomy Studio extension looks up specific folders. Each document of the folder is analyzed for its position in the taxonomy tree. The name of the folder in which the taxonomy tree is placed defines the name of the taxonomy tree and is visible as a root node in the taxonomy administration UI. First level taxonomies must be placed directly within the root folder. Taxonomies of subsequent levels can also be placed in subfolders.

The lookup folders for taxonomies and the strategy used to build the tree are configured in the Spring configuration file `component-taxonomies.xml`. The bean property

```
<property name="taxonomyFolders"
value="/Settings/Taxonomies/,Options/Taxonomies/" />
```

configures the folders that are used to find taxonomies. Relative paths will be concatenated with the sites root folder. The `taxonomyFolders` property is part of the `CMTaxonomyResolver` class which actually detects the trees and wraps the access to them through implementations of the interface `Taxonomy`. `CMTaxonomyResolver` implements the interface `TaxonomyResolver` so that it is possible to implement other taxonomy detection strategies.

How to implement a new taxonomy resolver strategy

The `CMTaxonomyResolver` implements the interface `TaxonomyResolver` and is injected to the `TaxonomyResource` so that a request for a taxonomy is made in *CoreMedia Studio*, the taxonomy resource instance looks up the corresponding

`Taxonomy` bean using the resolver instance. To change the resolver strategy, inject another instance of `TaxonomyResolver` to the `TaxonomyResource`.

How to implement a new taxonomy

If only the taxonomy build strategy will be changed, it is sufficient to keep the existing `CMTaxonomyStrategy`. And only modify the instance creation of `CMTaxonomy` and substitute it with an own implementation (for example a folder based taxonomy strategy).

How to configure the document properties used for semantic strategies

The document properties that are used for a semantic evaluation are configured in the file `semantic-service.xml`. The Spring configuration declares the abstract class `AbstractSemanticService` that new semantic service can extend from. The default properties used for a semantic suggestion search are:

- ➔ `title`
- ➔ `teaserTitle`
- ➔ `detailText`
- ➔ `teaserText`

How to implement a new suggestion/semantic strategy

To add a new semantic strategy to *Studio*, it is necessary to implement the corresponding strategy for it and add it to *CoreMedia Studio*.

A new semantic strategy can easily be created by implementing the interface `SemanticStrategy`. The result of a strategy is a `Suggestions` instance with several `Suggestion` instances in it. Each `Suggestion` instance must have a corresponding content instance in the repository whose content type matches that one used for the taxonomy. *Blueprint* uses `CMTaxonomy` documents for keywords of a taxonomy, so suggestions must be fed with these documents. Additionally, a float value `weight` can be set for each suggestion, describing how exactly the keyword matches from 0 to 1. After implementing the semantic strategy, the implementing class must be added to the Spring configuration, for example:

```
<customize:append id="semanticStrategyExamplesCustomizer"
bean="semanticServiceStrategies" order="1000">
  <list>
    <ref bean="myMatching"/>
  </list>
</customize:append>
```

Next the new suggestion strategy has to be added to *Studio*, so that is selectable in *CoreMedia Studio*. For that proceed as follows:

1. Open the ActionScript file `TaxonomyPreferencesBase.as`
2. Add a new key value for storing the strategy in the user preferences, for example

```
public static var TAXONOMY_MY_MATCHING_KEY:String = "myMatching";
```

Make sure that the constant value used here matched the Spring bean id of your suggestion strategy.

3. Add a new value to the taxonomy combo box in the preference dialog by adding the line

```
['Display name of My Suggestion Strategy',  
TAXONOMY_MY_MATCHING_KEY],
```

to method `getTaxonomyOptions()`. This will add the display name with the corresponding combo box item value to the taxonomy combo box.

4. Rebuild and restart *Studio* so that the changes take effect.

How to remove the OpenCalais suggestion strategy

If you want to disable the OpenCalais integration and remove the selection option from *Studio*, proceed as follows:

1. Remove the entry `<ref bean="semanticService"/>` from `taxonomies.xml`.
2. Remove the following line from the method `getTaxonomyOptions` of the `TaxonomyPreferencesBase.as` class:

```
[TaxonomyStudioPlugin.properties.INSTANCE.  
TaxonomyPreferences.value_semantic_opencalais_text,  
TAXONOMY_SEMANTIC_CALAIS_KEY]
```

3. In the same file as above, replace

```
DEFAULT_SUGGESTION_KEY:String = TAXONOMY_SEMANTIC_CALAIS_KEY;
```

with

```
DEFAULT_SUGGESTION_KEY = TAXONOMY_NAME_MATCHING_KEY;
```

How to add a site specific taxonomy

Adding a site specific taxonomy doesn't require any configuration effort. The logic how a site depending taxonomy tree is looked up can be found in class `CMTaxonomyResolver`.

To create a new site depending taxonomy proceed as follows:

1. Open *Studio* and select the folder `Options/Taxonomies/` from the library.
2. Create a new sub folder with the name of the new taxonomy.

The location for the new taxonomy has been created now.

3. To identify the type of taxonomy (such as `CMTaxonomy` or `CMLocTaxonomy`) you have to create at least one taxonomy document in the new folder.

Once the taxonomy has been set up, additional nodes can be created using the taxonomy manager. If the new taxonomy does not appear as new element in the column on the left, press the reload button. It ensures that the `CMTaxonomyResolver` rebuilds the list of available taxonomy trees. The new taxonomy is shown in the root column afterwards, include the site name it is created in.

Creating site specific taxonomies allows you to overwrite existing ones. For example you create a new taxonomy tree called `Subject` for site `x` and open an article that is located in a sub folder of site `x`, the regular `Subject` taxonomy property editor on the `Taxonomies` tab in *CoreMedia Studio* will access the `Subject` taxonomy of your new site, not the one that is located in the global `Settings` folders. The suggestions and the chooser dialog will also work in the new taxonomy tree.

How to configure the taxonomy property editor for a taxonomy

CoreMedia Blueprint comes with two types of taxonomies: `Subject` and `Location`. The name of the taxonomy matches the folder name they are located in, which is `/Settings/Taxonomies`. When the taxonomy property editor for a *Studio* form is configured, these IDs are passed to the property editor, for example

```
<taxonomy:taxonomyPropertyField propertyName="subjectTaxonomy"
taxonomyId="Subject"/>
  <taxonomy:taxonomyPropertyField itemId="locTaxonomyItemId"
    propertyName="locationTaxonomy"
    taxonomyId="Location"/>
```

As mentioned in the previous section, it is possible to overwrite the existing location or subject taxonomy with a site depending variant. In this case, it is *not* necessary to change the configuration for the property field. The taxonomy property editor will always try to identify the site depending taxonomy with the same name first. If this one is not found, the global taxonomy with the given id will be looked up and used instead.

How to configure access to the taxonomy administration

The taxonomy plugin uses the `configurations-rest-extension` module to load configuration values from a `Settings` document. The configuration document `TaxonomySettings` that contains the name of the user groups that are allowed to administrate taxonomies is located in the folder `/Settings/Options/Settings`. Additional configuration files with the same name can be put in the folder

Options/Settings (Relative paths will be concatenated with the root folder of the active site.). The entries of the files will be added to the existing configuration. Below the default taxonomy settings are shown.

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <StringListProperty Name="administrationGroups">
    <String>global-manager</String>
  </StringListProperty>
</Struct>
```

6.3 Website Management

Website management comprises different features. For example:

- Layout
- Navigation
- Search

6.3.1 Folder and User Rights Concept

It is good practice to organize the content of a content management system in a way that separates different types of content in different locations and to have user groups that attach role depending rights to these locations. This fits with CoreMedia access rights, which are assigned to groups and grant rights to folders and their content, including all sub folders, to all members of that group. See Section 3.16, “User Administration” in *CoreMedia Content Server Manual* for details about the CoreMedia rights system.

CoreMedia Blueprint comes with demo sites that provide a proposal on how to structure content in a folder hierarchy and how to organize user groups for different roles. A more fine grained folder and group configuration can easily be built upon this base. For details on site specific groups and roles have a look at [Groups and Rights Administration for Localized Content Management \[338\]](#) and for a set of predefined users for that groups and roles see [Appendix - Predefined Users \[484\]](#).

CoreMedia Blueprint distinguishes between the following types of content in the repository:

- **Content:** These are the "real" editorial contents like Articles, Images, Videos, and Products. They are created and edited by editorial users. In a multi-site environment editors are usually working on one of the available sites and they can only access that site's content.
- **Navigation and page structure:** These types represent the site's navigation structure - both the main navigation as well as the on-page navigation elements like collections or teasers linking to other pages. They are readable by every editorial user, but only the site manager group may maintain them.
- **Technical content types** like options, settings and configuration: These types provide values for drop down boxes in the editorial interface, like view types. They also bundle reusable sets of context settings, for example API keys for external Services. These types are readable by every editorial user but can only be created and edited by Administrators or other technical staff.
- **Client code:** Consists of Javascript and CSS and is maintained by technical editors.

*Different content types
for different uses*

CoreMedia Blueprint comes with a folder structure that simplifies groups and rights management in that way that users taking specific roles only get rights to those contents they are required to view or change. Most notably you will find a `/Sites` folder which contains several sites and several other folders which contain globally used content like global or default settings. For details on the structure of the `/Sites` folder have a look at [Section “Sites Structure” \[334\]](#).

Commonly used content is stored below dedicated folders directly at root level. Web resources like CSS or JavaScript is stored under `/Themes`. Global settings, options for editorial interfaces, and the like are stored under `/Settings`.

Site-Independent Groups

Along with the site specific groups which are described in [Groups and Rights Administration for Localized Content Management \[338\]](#) there are also groups representing roles for global permissions required by some of the predefined workflows. These workflows are especially dedicated to the publication process and are bound to the following roles:

composer-role

This site-independent group allows members to participate in a workflow as a composer, that is each member of this group may compose a change set for a publication workflow.

approver-role

This site-independent group allows members to participate in a workflow as an approver, that is each member of this group may perform approval operations within a publication workflow.

publisher-role

This site-independent group allows members to participate in a workflow as a publisher, that is each member of this group may publish the content items involved in a workflow.

For details on these groups and how to connect them to a LDAP server have a look at *CoreMedia Workflow Manual*.

6.3.2 Navigation and Contexts

Requirements

Websites are structured into different sections. These sections frequently form a tree hierarchy. For example, a news site might have a Sports section with a Basketball subsection. The website of a bank might have different sections for private

and institutional investors with the latter having subsections for public and private institutions.

Sections are also often called "navigation" or "context". Usually the sections of a site are displayed as a navigable hierarchy (a "navigation" or "site map"). The current location within the tree is often displayed as a "breadcrumb navigation".

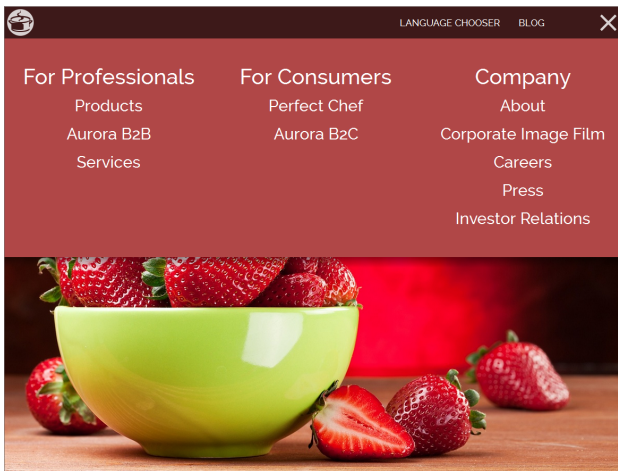


Figure 6.7. Navigation in the Perfect Chef Site

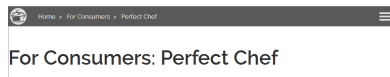


Figure 6.8. Bread-crumbs in the Corporate Blueprint Site

Additionally, efficient content management requires reuse of content in different contexts. For example, reuse of an article for a different section, a mobile site or a micro site should not require inefficient and error-prone copying of that article.

Solution

A site section (or "navigation" or "context") is represented by a content item of type `CMChannel` or `CMExternalChannel` which is a child of `CMChannel`. Sections span a tree hierarchy through the child relationships of `CMChannel#children`. If a `CMChannel` is referenced by a `CMSite` item it is considered a root channel, that is an entry into a channel hierarchy representing a website. The `CMChannel` content items fulfill the following purposes:

- **Hierarchy:** They form a hierarchy of site sections which can be displayed as a navigation, sitemap, or bread crumb. Each site consists of exactly one section tree.
- **Context:** They function as contexts for content. Content can be reused within different contexts in different layouts and visual appearance. For ex-

ample, an article's layout may differ in a company's blog section from its layout in the knowledge base.

- **Page:** Each `CMChannel` can be rendered as an overview page of the section it represents. Therefore, the `CMChannel` contains information about the page structure (the "grid") for this overview page and the pages generated when content items are displayed in the content of the `CMChannel`. For more information on how web pages are assembled in *Blueprint* also refer to the [Section 6.3.4, "Page Assembly" \[269\]](#) section.
- **Configuration:** `CMChannel` content items contain settings which configure various aspects of the site section they represent. Each `CMChannel` can override parent configuration by defining its own layout settings, content visibility, and other context settings. If for example, the "News" section of a site is configured for post-moderation of comments this configuration can be overwritten to premoderation in the subsection "News/Politics". For more information on settings see the section [Section 6.3.3, "Settings" \[268\]](#).

The context in which a content should be displayed is determined whenever a URL to the content is created. In a simple website with no content reuse all contents only have a single context and link building is very simple. For more complex scenarios *Blueprint* includes a `ContextStrategy` for the following purposes:

- Generate a list of the available contexts for a content (the `ContextFinder`).
- Determine the most appropriate context for the specific link to be built (the `ContextSelector`).

The `DefaultContextStrategy` in *Blueprint* uses a list of `ContextFinders` to retrieve all possible contexts for a content item and a single `ContextSelector` to determine the most appropriate one from the list.

The main `ContextFinder` in *CoreMedia Blueprint* is the `FolderPropertiesEvaluatingContextFinder`. Its logic to retrieve contexts is as follows:

1. Determine the folder of the content item.
2. Traverse the folder hierarchy starting from the folder in step 1 to the root folder looking for a content item of type `CMFolderProperties` named `_folderProperties`.
3. Return the contents of the linklist property `contexts` of the found `CMFolderProperties` document.

The `ContextSelector` in *CoreMedia Blueprint* is the `NearestContextSelector`. From the list of possible contexts for a content it selects the context closest to the current context.

6.3.3 Settings

Requirements

Editorial users must be able to adjust site behavior by editing content without the need to change the code base and redeploy the application. For example:

- Enable/disable comments for a certain section or the whole site.
- Set the number of dynamically determined related content items that are shown in an article detail view.
- Configure the refresh interval for content included from an external live source.

Administrative users must be able to adjust more technical settings through content, for example:

- Manage API keys for external services
- Image rendering settings
- Localization of message bundles

Solution

CoreMedia Blueprint uses Markup properties following the CoreMedia Struct XML grammar to store settings. Struct XML offers flexible ways to conveniently store typed key-value pairs where the keys are Strings and the values can be any of the following: String, Integer, Boolean, Link, Struct (allows for nested sub Structs). For more information on the Structs and CoreMedia Struct XML please see the chapter Structs in the [CoreMedia Unified API Developer Manual]

Settings can be defined on all content types inheriting from `CMLinkable`.

localSettings	
UI-Name	Local Settings
Description	The settings defined specifically on this CMLinkable.
linkedSettings	
UI-Name	Linked Settings
Description	A list of reusable CMSettings documents that contain a bundle of settings.

Table 6.10. Properties of CMLinkable for Settings Management

The local settings are easiest to edit. However, if you want to share common settings across multiple contents, you should spend the few extra steps to put them into a separate `Settings` document and add it to the linked settings in order to facilitate maintenance and ensure consistency. Some projects make use of settings quite extensively. Multiple `Settings` documents are a good instrument to structure settings of different aspects. You can still override single settings in the local settings, which have higher precedence.

The application also considers settings of the content's page context. If you declare a setting in a page, it is effective for all contents rendered in the context of this page. Settings are inherited down the page hierarchy, so especially settings of the root page are effective for the whole site, unless they are overridden in a subpage or a content.

For more detailed information and customization of the settings lookup strategy see [Section “The Settings Service” \[142\]](#) and the `SettingsService` related API documentation.

Settings as Java Resource Bundles

In a typical web application there is the need to separate text messages (such as form errors or link texts) from the rendering templates as well as rendering them according to a certain locale. The Spring framework provides a solution for these needs by the concepts of `org.springframework.context.MessageSource` for retrieving localized messages and by `org.springframework.web.servlet.LocaleResolver` for retrieving the current locale. Certain JSP tags such as `<form:error>` or `<spring:message>` are built on top of these concepts.

In *CoreMedia Blueprint*, localized messages are stored as settings in Structs as described above and can be accessed as `java.util.ResourceBundle` instances.

A handler interceptor (`com.coremedia.blueprint.cae.web.i18n.ResourceBundleInterceptor`) is used to make these content backed messages (as well as the current locale) available to the rendering engine: They are extracted from the content and passed to a special Spring `MessageSource`, the `RequestMessageSource` by storing it in the current request. As a consequence, using JSP tags like `<spring:message>`, `<form:error>` or `<fmt:message>` will transparently make use of these messages.

6.3.4 Page Assembly

Requirements

Requirements

For a good user experience a website should not layout each and every page in a different fancy manner but limit itself to a few carefully designed styles. For example, most pages consist of two columns of ratio 75/25, where the left column

shows the main content, and the right column provides some personalized recommendations.

In the best case an editor needs to care only for the content of a page, while the layout and collateral contents are added automatically, determined by the context of the content. However, there will always be some special pages, so the editors must be able to change the layout or the collateral contents. For example for a campaign page which features a new product they may omit the recommendations section and choose a simple one-column layout without any distracting features. In order to preserve an overall design consistency of the site, editors are not supposed to create completely new layouts. They can only choose from a predefined set.

Solution

CoreMedia Blueprint addresses these requirements with the concept of a page grid and placements.

The page grid does not handle overall common page features such as navigation elements, headers, footers and the like. Those are implemented by Page templates with special views. Neither does the page grid control the layout of collections on overview pages. This is implemented by `CMCollection` templates with special views and view types.

You can think of a page grid as a table which defines the layout of a page with different sections. Each section has a link to a symbol document which will later be used to associate content with the section. Technically, the layout of a page is defined in form of rows, columns and the ratio between them. A page grid contains no content and can be reused by different pages. So you might define three global page grids from which an editor can select one, for instance.

Page grid defines layout of a page

The content for the page grid on the other hand, is defined in a `CMChannel` document in so called placements, realized as link lists in structs. Each placement is associated with a specific position of the page grid through a link to a symbol document. The editor can add content to the placement, collections for example, which will be shown at the associated position of the page grid.

CMChannel contains content for page

Placements can also be shared between channels because a child inherits the placements of its parent. A prerequisite for inheritance is that the page grids of the parent and child page must have sections with the same name. For example, the parent channel has a two-column layout with the sections "main" and "sidebar". The child channel has a three-column layout with the sections "main", "sidebar" and "leftcolumn". For the placements this means:

Inheriting placements

- ➔ The child must fill a placement with content for the "leftcolumn" section, because the parent has no such section.

- ➔ The child will override the placement for the "main" section with its content. Inheritance makes no sense for the "main" section.
- ➔ The child does not need to declare a "sidebar" placement but can inherit the "sidebar" placement of the parent, even though it uses a different layout.

Before going into the implementation details of the page grid, you will see how to work with page grids in *CoreMedia Studio*.

Page grids in CoreMedia Studio

Editors can manage pages directly by editing the "placements" in the page grid in `CMChannel` documents (localized as `Page` in *CoreMedia Studio*). A placement is a specific area on a page such as the navigation bar, the main column or the right column. A `CMChannel` can inherit page grid placements of its parent channel. For example, the Sports/Football section of a site can inherit the right column from the Sports section. Editors can also choose to "lock" certain placements and thus prevent subchannels from overwriting them. Each page grid editor provides a combo box to choose between different layouts for a page. Depending on the selected layout, placement may inherit their content if the same placement is defined in the layout of the parent page.

Inheriting placements and locking

Each placement link list can configure a view type. The view type determines how the placement is rendered.

Layout of placement via view type

To define which placement view types are available for a page in some site, view type (`CMViewtype`) documents are placed in view type folders under a site-relative path or at global locations. The default paths are the site-relative path `Options/Viewtypes/` and the absolute path `/Settings/Options/Viewtypes/`. This can be configured via the application property `pagegrid.viewtype.paths`, which contains a comma-separated list of repository paths. Each path may start with a slash (`/`) to denote an absolute path or with a folder name to denote a path relative to a site root folder. When changing these values, please make sure that the existing view type documents are moved or copied to the new target location.

Web pages are represented in the CAE using the `com.coremedia.blueprint.com-mon.contentbeans.Page` object which consists of two elements: the content to be rendered and the context in which to render the content.

Pages where the content to be rendered is the same as the context (for example, section overview) display the page grid of the context. Pages where content items (such as Articles) are displayed within a context use display the context's page grid but replace the "main" placement with the content item.

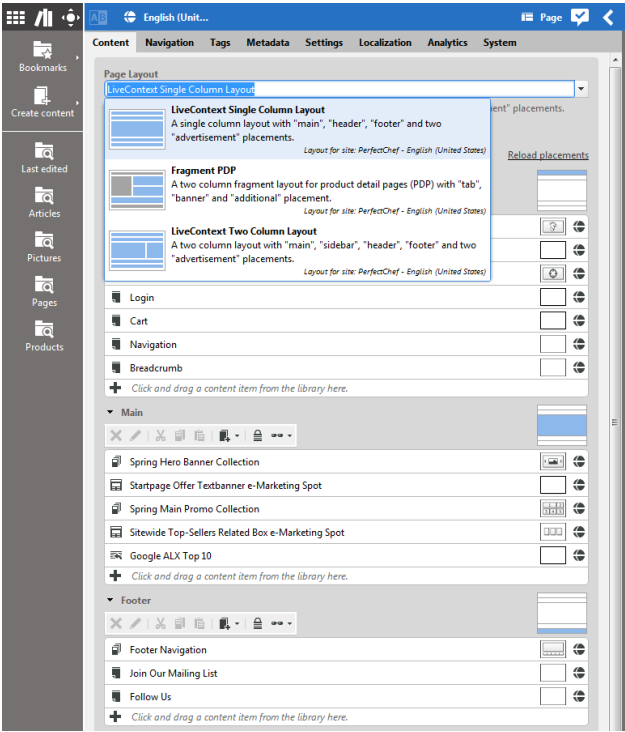


Figure 6.9. The page grid editor

Each placement has an icon symbolizing where it is located on the page. It contains a link list and several additional buttons on top of it. The order of the linked elements can be modified using drag and drop.

Icon to show location on page

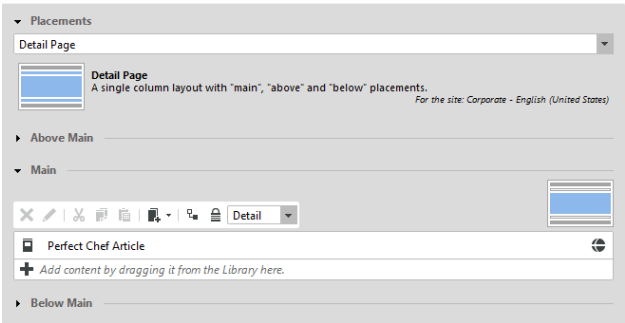


Figure 6.10. The main placement of a page

Instead of adding own content, a placement can inherit the linked content from a parent's page placement. If you inherit the content, you cannot edit the placement

Inheriting content from parent page

in the child page. You have to deactivate the "override" button to change the content of the placement.

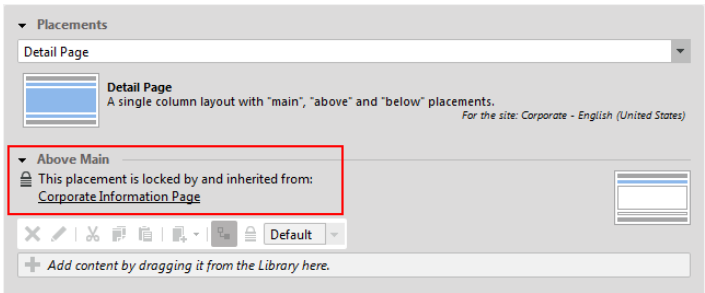


Figure 6.11. An inheriting placement

A placement can be locked using the "lock" button. In this case all child placements are not able to overwrite this placement with own content.

Locking placement

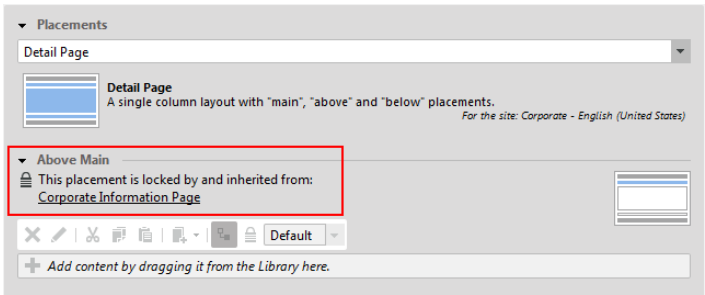


Figure 6.12. A locked placement

The page grid editor provides a combo box with predefined layouts to apply to the current page. After changing the layout, the *Studio* preview will immediately reflect the new page layout.

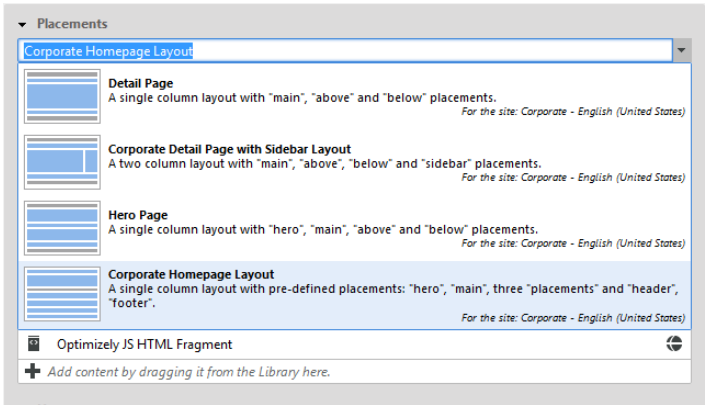


Figure 6.13. The layout chooser combo box

The layout of a parent page grid may be changed so that it does not fit anymore with the layout of a child page which inherits some settings. A child may use a three-column layout and inherit most of its content from its parent page that also uses a three-column layout. Then, the layout of the parent may be changed to another layout with a single column that doesn't contain any of the needed layout sections. The child configuration is invalid in this case and the user has to reconfigure all child pages.

Inconsistency between parent and child page grid

Currently there is no kind of detection for these cases in *Studio*, so the user has to check manually if the child configurations are still valid.

No check for inconsistency

How to configure a page grid editor

The Blueprint base module `bpbase-pagegrid-studio-plugin` provides an implementation of the page grid editor shown above through the config class `pageGridPropertyField` in the package `com.coremedia.blueprint.base.pagegrid.config`. In many cases, you can simply use this component in a document form by setting only the standard configuration attributes `bindTo`, `forceReadOnlyValueExpression`, and `propertyName`

If you want to adapt the columns shown in the link list editors for the individual section, you can also provide fields and columns using the attributes `fields` and `columns`, respectively. The semantics of these attributes match those of the `linkListPropertyField` component.

How to configure the layout location

Pages look up layouts from global and site specific folders. By default, the site specific page grid layout path will point to `Options/Settings/Pagegrid/Layouts` and the global one to `/Settings/Options/Settings/Pagegrid/Layouts`. This can be changed via the application property `pagegrid.layout.paths`, which contains a comma-separated list of repository paths. Each path may start

with a slash ('/') to denote an absolute path or with a folder name to denote a path relative to a site root folder. When changing these values, please make sure that the existing page layout documents are moved or copied to the new target location. Also, mind that when looking for the default page layout (see below), paths mentioned first take precedence, so it usually makes sense to start with site-relative paths and continue with absolute paths.

The default layout settings document `PagegridNavigation` must be present in at least one of the available layout folders. The page grid editor will show an error message if the document is not found.



If several layout folders are used, make sure that the layout settings documents have unique names.



How to configure a new layout

Every `CMSettings` document in a layout folder is recognized as a layout definition. The `settings` struct property defines a table layout with different sections. The struct defines two integer properties with the overall row and column count. The struct data may also contain two string properties `name` and `description`, which are used for the localization of page grid layout documents (see [section “How to localize page grid objects” \[279\]](#)).

The `items` property contains a list of substructs, each defining a section of the page grid. The order in which the sections appear in the struct list matches the order in which the link lists of the individual sections are shown by the page grid editor.

The sections are represented by `CMSymbol` documents. The layout definition is inspired by the HTML table model, even though *CoreMedia Blueprint's* default templates do not render page grids as HTML tables but with CSS means. The sections support the following attributes:

- `col`: The column number where the section is placed or, if the `colspan` attribute is set, the column number of the leftmost part of the section.
- `row`: The row number where the section is placed or, if the `rowspan` attribute is set, the row number of the topmost part of the section.
- `colspan`: The number of columns spanned by the section.
- `rowspan`: The number of rows spanned by the section.
- `width`: The width of this section in percent of the total width.
- `height`: The height of this section in percent of the total height.

The `col`, `row` and `rowspan` attributes of the section must match the grid layout defined by the `colCount` and `colRow` attributes (see [Example 6.1, “Pagegrid example definition” \[276\]](#)). That is, when `colCount` and `colRow` are “3” and “4”, for example, then you have 12 cells in the page grid table layout which must all be filled by the sections. No cell can be left empty, and no section can overlap with other sections.

The height attribute is only used for the preview of the layout in the page form. It has no impact on the delivered website.

The default `PagegridNavigation` layout settings document with a 75%/25% two column layout looks as follows:

Example 6.1. Pagegrid example definition

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <IntProperty Name="colCount">2</IntProperty>
  <IntProperty Name="rowCount">1</IntProperty>
  <StructListProperty Name="items">
    <Struct>
      <LinkProperty Name="section"
xlink:href="coremedia:///cap/content/550"
LinkType="coremedia:///cap/contenttype/CMSymbol"/>
      <IntProperty Name="row">1</IntProperty>
      <IntProperty Name="col">1</IntProperty>
      <IntProperty Name="height">100</IntProperty>
      <IntProperty Name="width">75</IntProperty>
      <IntProperty Name="colspan">1</IntProperty>
    </Struct>
    <Struct>
      <LinkProperty Name="section"
xlink:href="coremedia:///cap/content/544"
LinkType="coremedia:///cap/contenttype/CMSymbol"/>
      <IntProperty Name="row">1</IntProperty>
      <IntProperty Name="col">2</IntProperty>
      <IntProperty Name="height">100</IntProperty>
      <IntProperty Name="width">25</IntProperty>
      <IntProperty Name="colspan">1</IntProperty>
    </Struct>
  </StructListProperty>
  <StringProperty Name="name">2-Column Layout (75%,
25%)</StringProperty>
  <StringProperty Name="description">Two column layout with main
and sidebar sections</StringProperty>
</Struct>
```

The main content of a document will always be rendered into the `main` section of a layout. Therefore, every layout must define a `main` section.



How to configure a read-only placement

The page grid layout definition provides the possibility to declare a read-only section. The Boolean property “editable” has to be declared for the `struct` element of the corresponding section, for example:

```
<Struct>
  <LinkProperty Name="section"
xlink:href="coremedia:///cap/content/120"
LinkType="coremedia:///cap/contenttype/CMSymbol"/>
  <IntProperty Name="row">2</IntProperty>
  <IntProperty Name="col">1</IntProperty>
  <IntProperty Name="colspan">1</IntProperty>
  <IntProperty Name="height">75</IntProperty>
  <IntProperty Name="width">25</IntProperty>
  <BooleanProperty Name="editable">false</BooleanProperty>
</Struct>
```

The section that matches the given symbol will be shown as disabled in *Studio*. The matching placements will not appear in the editor.

How to populate a page grid with content

Page grids are defined in the struct property `CMNavigation.placement` of a channel. Such structs are typically created using the page grid editor shown above. Example:

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <StructListProperty Name="placements">
    <Struct>
      <LinkProperty Name="section"
LinkType="coremedia:///cap/contenttype/CMSymbol"
xlink:href="coremedia:///cap/content/550"/>
      <LinkProperty Name="viewtype"
LinkType="coremedia:///cap/contenttype/CMViewtype"
xlink:href="coremedia:///cap/content/1784"/>
      <LinkListProperty Name="items"
LinkType="coremedia:///cap/contenttype/CMArticle">
        <Link xlink:href="coremedia:///cap/content/134"/>
        <Link xlink:href="coremedia:///cap/content/498"/>
      </LinkListProperty>
    </Struct>
    <Struct>
      <LinkProperty Name="section"
LinkType="coremedia:///cap/contenttype/CMSymbol"
xlink:href="coremedia:///cap/content/544"/>
      <LinkListProperty Name="items"
LinkType="coremedia:///cap/contenttype/CMArticle">
        <Link xlink:href="coremedia:///cap/content/776"/>
      </LinkListProperty>
    </Struct>
  </StructListProperty>
  <StructProperty Name="placements_2">
    <Struct>
      <LinkProperty Name="layout"
LinkType="coremedia:///cap/contenttype/CMLayout"
xlink:href="coremedia:///cap/content/3488"/>
    </Struct>
  </StructProperty>
</Struct>
```

A placement struct contains a list of section structs `placements`. The `placements_2` struct contains another struct, `placements` and a link property `layout`, which determines the layout for this channel.

The `placements` struct property consists of substructs for the single placements, each of which refers to a section and lists its contents in the `items` property. Additionally, each placement can declare a view type.

Layouts and placements are connected by the section documents. Let's assume you have two sections, "main" and "sidebar". Your channel declares some latest news for the main section and some personalized recommendations for the sidebar. The layout definition consists of one row with two columns, the left column refers to the "main" section, the right column refers to the "sidebar". This will make your channel be rendered with the main content left and the recommendations on the right. If you don't like it, you can simply choose another layout, for example with a different width ratio of the columns or with the sidebar left to the main section.

The rendering of a page grid is layout-driven, because the sections of the table-like layout model must be passed to the template in an order which is suitable for the output format (usually HTML). CoreMedia Blueprint's web application processes a page grid as follows:

1. The `PageGridServiceImpl` determines the layout document of the channel. If there is no layout link in the `placements_2` struct, a fallback document `PagegridNavigation` is used. This name can be configured by setting the application property `pagegrid.layout.defaultName`. The fallback layout document can be located in any of the configured layout folders (see "layout locations"), usually it will be located under the site relative path `Options/Settings/Pagegrid/Layouts`. The layout definition is evaluated and modeled by a `ContentBackedStyleGrid`.
2. The `PageGridServiceImpl` collects the placements of the channel itself and the parent channel hierarchy. The precedence is obvious, for example a channel's own placement for a section ("sidebar" for instance) overrides an ancestor's placement for that section.
3. Both layout and placements are composed in a `ContentBackedPageGrid` which is the backing data for a `PageGridImpl`. `PageGridImpl` implements the `PageGrid` interface and prepares the data of the `ContentBackedPageGrid` for access by the templates. Basically
 - ➔ it wraps the content of the placements into content beans,
 - ➔ it arranges the placements in rows and columns, according to the layout
 - ➔ it replaces the channel's main placement with the requested content.

Blueprint's default templates (namely `PageGrid.jsp`) do not render page grids as HTML tables but as nested `<div>` elements and suitable CSS styles. The beginning of a rendered page grid looks like this:

```
<div id="row1" class="row">
  <div id="main" class="col1 column collof2 width67">
```

The outer `<div>` elements represent the rows of the page grid, the inner `<div>` elements represent the columns. The ids of the rows are generated by the template as an enumeration. The ids of the columns are the section names of the placements. The column `<div>` elements are rendered with several class attributes:

- `column`: A general attribute for column `<div>` elements
- `col1`: The absolute index of the column in its row
- `collof2`: The `colspan` of this column (1) and the absolute number of columns of the page grid (2)
- `width67`: The relative width of this column

You can use these attributes to define appropriate styles for the columns. *CoreMedia Blueprint*'s default CSS provides styles which reflect the width ratios of some typical multi-column layouts. You find them in the document `/Themes/basic/css/basic.css` in the content repository where you can enhance or adapt them to your needs.

In the inner `<div>` elements the placements are included, and their section names determine the views. For example a "sidebar" placement is included by the `PageGridPlacement.sidebar.jsp` template.

How to localize page grid objects

To localize a layout name, create a resource bundle entry with the key `<layoutname>_text` in the resource bundle `PageGridLayouts_properties`, where `<layoutname>` is the name of the layout document or, preferably, the name property of the settings struct of the layout. Similarly, a layout description can be localized with entries of the form `<layoutname>_description`. If no corresponding resource bundle entries are found, the `description` property of the settings struct of the layout is used. If that property is empty, too, the name is used as the description. The resource bundle is available in the package `com.coremedia.blueprint.base.pagegrid` of module `bpbase-pagegrid-studio-plugin`.

For the purposes of localization, placements are treated as pseudo-properties and localized according to the standard rules for content properties as described in the [Studio Developer Manual]. The name of the pseudo-property is `<structname>-<placementname>`, where `<structname>` is the name of the struct property storing the page grid and `<placementname>` is the name of the section document. For example, a placement with the name `main` that is referred from the standard page grid struct `placement` of a `CMChannel` document would obtain its localization using the key `CMChannel_placement-main_text`. You can add localization

entries to the resource bundle `BlueprintDocumentTypes_properties` of module `blueprint-forms`, which is applied to the built-in resource bundle `ContentTypes_properties` at runtime.

To localize a view type name or a view description, you can add a property `<viewtypename>_text` or `<viewtypename>_description` to the bundle `Viewtypes_properties`. Here `<viewtypename>` is the name of the view type document or, preferably, the string stored in its `layout` property. Because view types are also used in other contexts, this bundle has been placed in the package `com.coremedia.blueprint.base.components.viewtypes` of module `bp-base-studio-components`.

CoreMedia Blueprint defines three resource bundles `BlueprintPageGridLayouts_properties`, `BlueprintPlacements_properties`, and `BlueprintViewtypes_properties`. Entries of these bundles are copied to the bundles described above, providing a convenient way to add custom entries.

6.3.5 Overwriting Product Teaser Images

Requirements

You have put a product teaser on your home page, which is displayed with the default product image coming from the commerce system but you want to highlight that teaser by changing its default image to a more engaging one.

Solution

CoreMedia Digital Experience Platform 8 allows you to either use the content from the e-Commerce database or overwrite this image with your own image in the `Teaser` content type.

6.3.6 Content Lists

Requirements

Websites frequently display content items that share certain characteristics as lists, for example, the top stories of the day, the latest press releases, the best rated articles or the recommended products. Some of these lists are managed editorially while others should be compiled dynamically by business rules defined by editors. It is a common requirement to reuse these content lists across different web pages and use common functionality to place lists on pages and assign different layouts to lists.

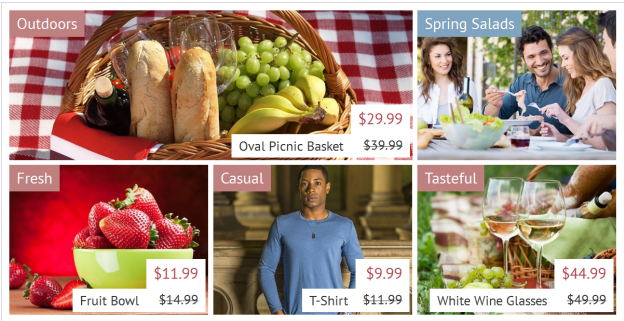


Figure 6.14. Teaser collection with prices

Solution

CoreMedia Blueprint defines different content types for lists of content which differ in how they determine the content items. Leveraging CoreMedia's object oriented content modeling these lists can reuse view templates and can be placed interchangeably on web pages.

Table 6.11. Collection Types in CoreMedia Blueprint

Type	Purpose
CMCollection	A common base type for lists, which all other list types extend. It provides functionality for editorially managed lists.
CMGallery	A distinct content type for lists of CMMedia content items which should be displayed as a gallery.
CMQueryList	Dynamic lists that are based on content metadata, such as "latest 5 articles in sport".
CMSelectionRules (part of Adaptive Personalization)	Dynamic lists that are based on context information with rules defined by editorial users, such as "if a visitor is interested in notebooks, display this product, otherwise display something else."
CMP13NSearch (part of Adaptive Personalization)	Dynamic lists based on content metadata and context information, such as "display list of articles matching the current visitor's bookmarked taxonomies."
ESDynamicList (part of Elastic Social)	Dynamic lists that are based on Elastic Social metadata, such as "5 best rated articles in news."
CMALXPageList (part of Analytics)	Dynamic lists that are bases on analytics data, such as "10 most viewed articles in business."

6.3.7 View Types

Requirements

A common pattern for CoreMedia projects is to reuse content and display the same content item on various pages in different layouts and view variants. A content list, for example, could be rendered as simple bulletin list or as a list of teasers with thumbnails. Similarly, an article can be displayed in a default ("full") view or as a teaser.

Usually the rendering layer decides what view should be applied to a content item in different use cases. For example, the view rendering results of a search on the website could use the `asListItem` view to render the found items.

Editors still need a varying degree of control to influence the visual appearance of content in specific cases. They might want to decide whether a list of content items should be displayed as a teaser list or a collapsible accordion on a page, for example.

Solution

A dedicated content type called `CMViewtype` is available that can be associated with all `CMLinkable` content types. During view lookup a special `com.coremedia.objectserver.view.RenderNodeDecorator`, the `ViewTypeRenderNodeDecorator`, augments the view name by the `layout` property of the view type referenced by the content item.

The `BlueprintViewLookupTraversal` then evaluates this special view name and falls back to the default view name without the view type if the view could not be resolved.

In the example above the template responsible for rendering search results would include all found content with the `asListItem` view. If the content is of type `CMArticle` there would be a lookup for a `CMArticle.asListItem.jsp` (among others in the content object's type hierarchy, see section [Views] in the [Content Application Developer Manual] for more CoreMedia's object oriented view dispatching). If the article has a view type assigned (such as *breakingnews*) there would be a lookup for `CMArticle.asListItem[breakingnews].jsp` before falling back to `CMArticle.asListItem.jsp`. This allows for very fine grained editorially driven layout selection for any created content.

Selecting a view type in CoreMedia Studio

You can use the view type selector which is associated with the view type property to select a specific view type for a document, a collection for instance. The view type selector is implemented as a combo box providing an icon preview and a description text about the view type. View types can be defined globally or site specific. If the view type item is configured for a site, the name of the site is also displayed in the combo box item.

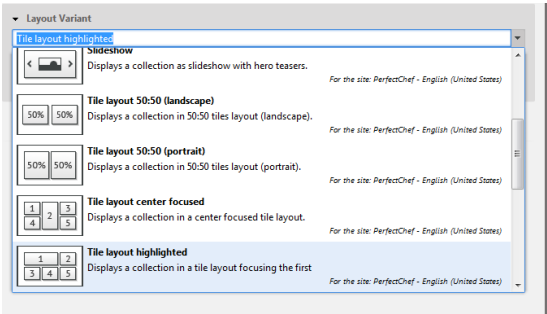


Figure 6.15. Layout Variant selector

The view type selector combo box of the image gallery document.

How to configure a view type selector

There are several document forms that include the view type selector form EXML. The `ViewTypeSelectorForm` bundles the view type selector combo box and its configuration parameters. Basically the parameters `paths` and `allowedDoctypes` define which items are shown in the combo box. The combo box assumes that each of the items (`CMViewtype` here) has a property `icon` that contains the thumbnail view of the view type.

```
<bpforms:viewTypeSelectorForm propertyName="viewtype"
  paths="{['/Settings/Options/Viewtypes/CMTeasable',
  'Options/Viewtypes/CMTeasable']}"
  withEmptySelection="true"
  allowedDoctypes="CMViewtype"/>
```

In this example all `CMViewtype` documents of the folders `/Settings/Options/Viewtypes/CMTeasable` and `Options/Viewtypes/CMTeasable` (site depending) are shown in the view type selector combo.

How to localize view types for the view type selector

The view type selector displays two fields of a view type: The name (which is the name of the document in the repository) and the `description` property. These string can be localized as described earlier in [section “How to localize page grid objects” \[279\]](#).

6.3.8 CMS Catalog

Requirements

Some companies do not run an online store. They do not need a fully featured shopping system. Nonetheless, they want to promote some products on their corporate site.

Solution

CoreMedia Digital Experience Platform 8 provides the *CMS Catalog*, an implementation of the *e-Commerce API* (known from the WebSphere Commerce Integration), which is backed only by the CMS and does not need a third-party commerce system. It allows to maintain a smaller number of products and categories for presentation on the website. It does not support shopping features like availability or payment. The *CMS Catalog* is based on *Blueprint* features. It is already integrated in the `Corporate` extension, so you can use it out of the box.

Table 6.12. CMS Catalog: Maven parent modules

Maven Module	Description
<code>com.coremedia.blueprint.base:bpbse-commerce</code>	Contains the <i>e-Commerce API</i> implementation for the cms. The implementation is content type independent.
<code>com.coremedia.blueprint:ecommerce</code>	Contains the content types, content beans and the studio catalog component.
<code>com.coremedia.blueprint:corporate</code>	Example usage of the catalog in the corporate page.

Content Types

In the *CMS Catalog* products and categories are modeled as content. There are two new content types, `CMProduct` and `CMCategory`, which extend the well known *Blueprint* document types `CMTeasable` and `CMChannel`, respectively. So you can seamlessly integrate categories into your navigation hierarchy and place products on your pages, just like any other content. In order to activate the new content types you have to add a Maven runtime dependency on the `catalog-doctypes` module to your Content Server components.

Content Beans

The modules `catalog-contentbeans-api` and `catalog-contentbeans-lib` provide content beans for `CMProduct` and `CMCategory`. The content beans integrate into the class hierarchy according to their content types, that is they extend `CMTeasable` and `CMChannel`, respectively. The content beans do not implement the *e-Commerce API* interfaces `Product` and `Category`, though. Instead, they provide delegates via `getProduct` and `getCategory` methods. While this may look inconvenient at first glance, it has some advantages concerning flexibility:

- The content bean interfaces remain independent of future changes in the *e-Commerce API*.
- You have better control over the view lookup by explicitly including the content bean or the delegate.

Configuration

First, you need three settings in the root channel to activate a CMS Catalog for your site. *Blueprint Base* provides a commerce connection named `cms1` which is backed by the content repository. You can activate this connection by the `livecontext.connectionId` setting. Moreover, your catalog needs a name, which is specified by the `livecontext.store.name` setting. Finally, your catalog needs a root category, which is specified by the `livecontext.rootCategory` setting.

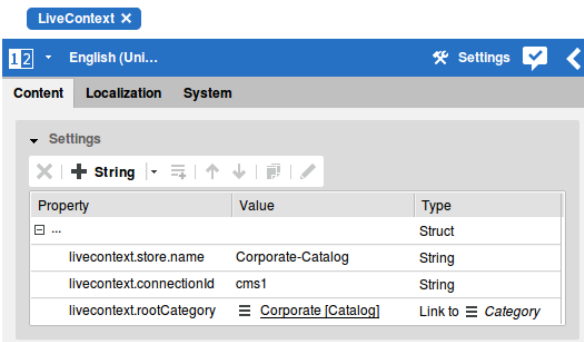


Figure 6.16. CMS Catalog Settings

Although the catalog indicator is a `CMCategory` document, it does not represent a category but serves only as a technical container for the actual top categories (see *e-Commerce API*, `CatalogService#findTopCategories`). The concept resembles the site indicator, which is the point of entry to the navigation without being part of it.

In a multi-site project sites may have different commerce connections. In order to make `Commerce#getCurrentConnection` work correctly regarding to the site a particular request refers to, you need to declare a Maven runtime dependency on the `bpbases-ec-cms-component` module and import some magic into the CAE Spring configuration:

```
<import
resource="classpath:/com/coremedia/blueprint/ecommerce/cae/ec-cae-lib.xml"/>
```

While the product → category relation is modeled explicitly with the `contexts` link list, the reverse relation uses the search engine. Therefore, you need to extend the `contentfeeder` component with some Spring configuration from the `bpbases-ec-cms-contentfeeder-lib` module:

```
<import
resource="classpath:/framework/spring/bpbases-ec-cms-contentfeeder.xml"/>
```

Templating

You can use both, `Product` or `CMProduct` templates. You can also use a mixture of both for different views or fallback to `CMTeasable` templates for views that do not involve `CMProduct` specific features.

Using `Product` templates you can easily switch to a third-party e-Commerce system later, since the interface remains the same. Otherwise you are more flexible with `CMProduct` templates:

- You can easily enhance the `CMProduct` content type and interface and access the new features immediately.
- You benefit from all the inherited features (like multi language) and fallback capabilities along the content type driven interface hierarchy.
- You can easily switch from `CMProduct` to `Product` just by calling `CMProduct#getProduct` anywhere you need a `Product` object. The reverse direction is more cumbersome.

6.3.9 Teaser Management

Requirements

Most websites present short content snippets as "teasers" on various pages. Content and layout for teasers should be flexible but manageable with minimum effort.

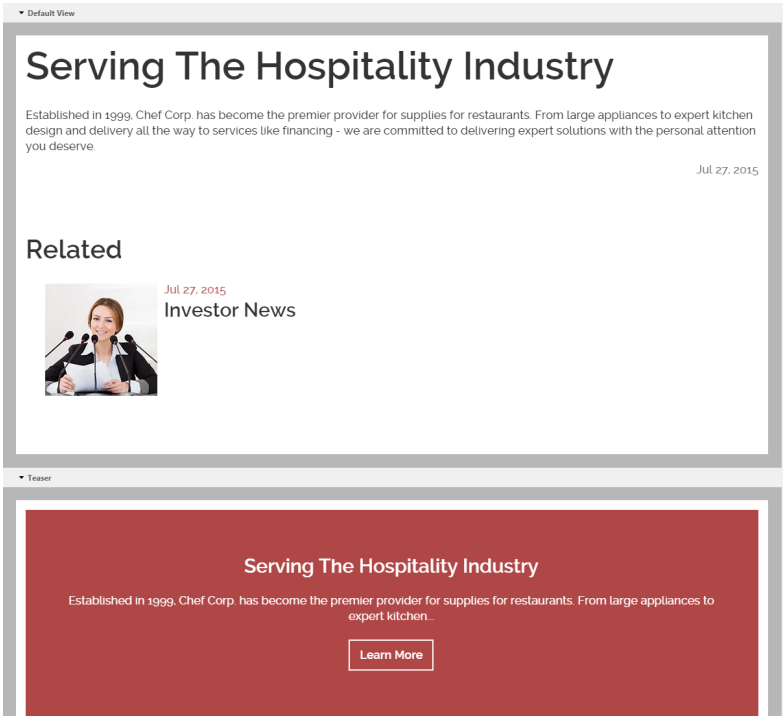


Figure 6.17. Default view and teaser view of an Article

It is a common requirement to automatically derive abbreviated content teasers without the need to duplicate any content items. In some cases, editors wish to create distinct teasers for a content item that don't reuse any information from that item.

Example: An editor wants to point to an article using a specific image that is not part of that article. Or: An editor wants to promote an article on a page with a teaser that is not the default teaser (using different text, image, or layout).

Solution

In *CoreMedia Blueprint* all content types for content and pages extend from the abstract content type `CMTeasable`. It defines common properties and business rules which provide all types inheriting from `CMTeasable` with a default behavior when displayed as a teaser.

Type	Purpose
teaserTitle	The title of the content item when displayed as a teaser.

Table 6.13. Properties of `CMTeasable`

Type	Purpose
teaserText	The text of the content item when displayed as a teaser.

Fallbacks to automatically display the shorter teaser variant of properties are implemented in the content bean implementation for `CMTeasable`. For example, the `teaserText` of a content reverts to the `detailText` if no `teaserText` has been entered by an author.

For distinct teasers *CoreMedia Blueprint* includes a `CMTeaser` content type that can be used for this purpose. It provides all properties required to display a teaser and can be linked to the content that it promotes. Teasers without a link are also supported to create non-interactive brand promotions etc.

6.3.10 Dynamic Templating

Requirements

In order to quickly implement microsites, campaigns, or specialized channels with unique template requirements, templates can be updated without interrupting the service or requiring a redeployment of the application.

Solution

Views can be implemented as FreeMarker templates and uploaded to the Content Repository in a container file, preferably a JAR. For details, consult the "Loading Templates from the Content Repository" chapter in the [CAE Developer Manual].

Create the archive containing the templates

A template set archive, preferably a JAR file, can contain FreeMarker templates which must be located under the path: `/META-INF/resources/WEB-INF/templates/siteName/packageName/`

The easiest way to create the JAR is to create a new Maven module with a POM like this one:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example.groupId</groupId>
  <artifactId>templates</artifactId>
  <version>--insert version here--</version>
  <packaging>jar</packaging>
  <description>
    CAE templates to be uploaded to a CMTemplateSet document in
    /Themes/*my.package*/templates/ with name
```



```
*my.package*-templates.jar.  
  
    Use the *my.package* as a reference in a Page's  
    "viewRepositoryNames" settings (list of strings).  
</description>  
  
<build>  
  <plugins>  
    <plugin>  
      <groupId>org.apache.maven.plugins</groupId>  
      <artifactId>maven-jar-plugin</artifactId>  
      <version>2.4</version>  
      <configuration>  
        <archive>  
          <addMavenDescriptor>>true</addMavenDescriptor>  
        </archive>  
      </configuration>  
    </plugin>  
  </plugins>  
</build>  
</project>
```

Put your templates below the path `src/main/resources/META-INF/resources/WEB-INF/templates/--themeName--/--packageName--/`, for example `src/main/resources/META-INF/resources/WEB-INF/templates/corporate/com.coremedia.blueprint.common.content beans/Page.ftl`

Upload the template set

CoreMedia Blueprint provides the content type `Template Set` (`CMTemplateSet`) which is used for this purpose. Create a document of type `Template Set` in folder `/Themes/--themeName--/templates` and upload the JAR to its `archive` property. Its name is significant and is used to reference template sets from channel settings, as explained see below.

Table 6.14. Properties of *CMTemplateSet*

Name	Description
description	A description of the purpose / contents of the code.
archive	blob property that contains the archive (preferably a JAR) that contains the templates.

Add the template set to a page

A Page context can be configured to add additional template sets to all pages rendered in its context. The names of additional template sets are configured in a string list setting `viewRepositoryNames` of a Page. Like all settings, a Page will inherit this list of names from its parent context, if it is not set. See [Section 6.3.11, “View Repositories” \[290\]](#) for more details.



The CAE will resolve view repository names automatically according to the pre-defined name pattern. For instance, if a Page sets its `viewRepositoryNames` to the list `["christmas", "campaigns"]`, each page rendered in this context will use templates implemented in the Template Sets `/Themes/christmas/templates/christmas-templates.jar` and `/Themes/campaigns/templates/campaigns-templates.jar` before falling back to the default templates defined for the web application.

6.3.11 View Repositories

Requirements

A CoreMedia deployment can host multiple sites which frequently differ in layout and functionality. It is a common requirement to use different view templates for those sites but still be able to define reused templates across sites flexibly.

Solution

The *CoreMedia* CAE offers a very flexible view selection mechanism by providing the `ViewRepositoryNameProvider` and `ViewRepositoryProvider` abstraction (see [Content Application Developer Manual], chapter "Views").

CoreMedia Blueprint offers the `BlueprintViewRepositoryNameProvider` implementation which for each lookup of model and view generates a list of view repository names to query. The list is created based on

- the specific view repository names defined in the String list setting `viewRepositoryNames` of the navigation context of the provided model,
- the view repository names defined via Spring in the property `commonViewRepositoryNames` on the `BlueprintViewRepositoryNameProvider` Java bean.

This allows for more fine-grained control of the used view repositories as view repositories can be configured not only specific for a site but also for each site section.

CoreMedia Blueprint uses the standard CAE `TemplateViewRepositoryProvider` to create from the list of view repository names the list of actual view repositories to query. *CoreMedia Blueprint* configures the following `templateLocationPatterns` for the `TemplateViewRepositoryProvider`:

- `jar:id:contentproperty:/Themes/%1$s/templates/%1$s-templates.jar/archive!/META-INF/resources/WEB-INF/templates/%1$s`

- ➔ `jar:id:contentproperty:/Themes/%1$s/templates/%1$s-templates.jar/archive!/META-INF/resources/WEB-INF/templates/sites/%1$s`
- ➔ `/WEB-INF/templates/sites/%s`
- ➔ `/WEB-INF/templates/%s`

Example: For a content of the corporate site the `BlueprintViewRepositoryNameProvider` yields the view repository names "corporate". The `TemplateViewRepositoryProvider` would then return the following view repositories which are queried for the responsible view:

- ➔ A FreeMarker template view repository in the CMS located in the `/Themes/corporate/templates/corporate-templates.jar` (a `CMTemplateSet`) content item's blob property archive
- ➔ A Freemarker or JSP file system view repository below `/WEB-INF/templates/sites/corporate`
- ➔ A Freemarker or JSP file system view repository below `/WEB-INF/templates/corporate`

6.3.12 Client Code Delivery

Requirements

Client code such as JavaScript and CSS is changing more rapidly than JSP templates and back-end business rules. To deliver JS and CSS changes conveniently it is a common pattern to consider those as content and use the common editorial workflow (create, approve, publish) to deploy these to the live environment.

Solution

CoreMedia Blueprint provides the content types `CMCSS` and `CMJavaScript` which both inherit from the common super type `CMAbstractCode`.

Name	Description
description	A description of the purpose / contents of the code.
code	The code stored in a CoreMedia XML property following the CoreMedia RichText schema. This allows for embedding images directly in a code fragment and enables quick fixes of client code in the standard CoreMedia editing tools.

Table 6.15. Client Code - Properties of *CMAbstractCode*

Name	Description
disableCompress	Prevents the CAE from compressing the code. Setting this flag is recommended if the code is either already compressed or if it is not compatible with the compression engine. The <i>CSS Importer</i> and the <i>coremedia-webresource-content-maven-plugin</i> automatically set this flag if the compatibility test fails or if the file extension is <code>.min.js</code> or <code>.min.css</code> .
include	Other code elements that should be deployed together with this one.
dataUrl	An (optional) URL of the code on an external system. Allows to also manage all code included from third-party servers as if it was part of the CoreMedia repository.

Client code is associated with site sections. `CMNavigation` content items contain references to the CSS and JavaScript items to be used within the section. Child sections inherit code from their parent. They can extend it to refine their section layout. This enables editorial users to quickly associate new design to sections that stand out from the rest of the page, or even roll out a site wide face lift without having to redeploy the application itself.

Table 6.16. Client Code - Properties of `CMNavigation`

Name	Description
javaScript	The <code>CMJavaScript</code> scripts used within the context.
css	The <code>CMCSS</code> style sheets used within the context.

Additional resources for preview

Additional `CSS` and `JavaScript` can be added to sites for use in *CoreMedia Studio* and the embedded preview. `CSS` will be included in `Page.head.ftl` and `JavaScript` in `Page.bodyEnd.ftl` after the regular web resources.

The settings are organized as linklist properties. The name of the `linklist` for `CSS` itself must be "previewCss" and "previewJs" for javascript. The settings must be attached to the root channel of a site.

Settings to add resources for preview

Web Performance Optimization

Besides the concepts for managing and deploying client code from within the content repository, *CoreMedia Blueprint* also features mechanisms to both speed up site loading and reduce request overhead during the delivery of web resources.

Requirements

Reducing the overhead of both client request count and data transfer sizes for client codes and web resources such as JavaScript and/or CSS.

Solution

- Minification of client codes: In order to reduce the data transfer to the client, JavaScript and CSS files are usually minified, meaning, that all unnecessary characters are removed from the source code. This results in smaller files, hence reduces the amount of data that needs to be transferred to the client. This can especially be useful for mobile clients. For the CAE, the minification of web resources is turned on by default. Each needed source file is processed by a minifier. The minifier strips all comments, whitespaces and any other unwanted information from the source code and compresses it. The ideal result will be a source file with just a single line of code.
- Merging of client codes: *CoreMedia Blueprint* also offers a merging process which compresses all JavaScript and CSS files into a single one each. The merging, if turned on, immediately follows the minification step. Client codes are also merged by default. As a matter of fact, both features cannot be turned on or of separately.

The process of minification and merging only applies to source files, that don't have a set IE Expression or Data URL property. If an IE Expression or Data URL is set, the file will be skipped in both process steps and result in each file rendered separately into the source code of the page.



Configure minification and merging

For debugging purposes during the development, it might come in handy to disable the minification and merging feature. You do that by turning on the `cae.developer.mode` property switch, either provided with a standard property file, or via a Maven switch. Inside the `cae-preview-webapp` module, all you have to do is to start the preview CAE web application locally using the Maven Tomcat plugin.

6.3.13 Managing End User Interactions

Requirements

For a truly engaging experience website visitors need to be able to interact with your website. Interactions can reach from basic ways to search content, register and give feedback to enabling user-to-user communication and facilitating business processes such as product registration and customer self care.

End user interactions should be configurable in the editorial interface by non-technical users in the editorial interface of the system. It should, for example, be

possible to place interaction components such as Login and Search buttons on pages just like any other content, configure layout and business rules etc.

Solution

For the Blueprint website, the term "action" denotes a functionality that enables users to interact with the website.

Examples:

- ➔ Search: The "search" action lets user to enter a query into a form field. After processing the search, a search result is displayed to the user.
- ➔ Login: This action can be used by users to login to the website by adding user name and password credentials. A successful login changes the state web application's state for the user and offers him additional actions such as editing his user profile.

From an editor's perspective, all actions are represented by content objects of type `CMAction`. This enables an editor to add an action content to a page, for example by inserting it to the navigation `linklist` property. When rendering the page, this action object is rendered by a certain template that (for example) renders a search form. The submitted form data (the query, for instance) is received by a handler that does some processing (passing the query to the search engine, for instance) and that provides a model containing the search action result.

This section demonstrates the steps necessary to add new actions to *CoreMedia Blueprint*. It also helps to understand the currently available actions.

Standard Actions

As stated above, all actions are represented as `CMAction` contents in the repository. These contents can be used as placeholders in terms of the "substitution" mechanism described in the [CAE Developer Manual]. An example for adding a new action: Consider an action where users can submit their email addresses in order to receive a newsletter.

1. Create a bean that represents the subscription form and add an adequate template.

```
public class SubscriptionForm {
    public String email;

    public void setEmail(String email) {
        this.email = email;
    }

    public String getEmail() {
        return email;
    }
}
```

```
}
```

SubscriptionForm.asTeaser.jsp

```
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form"%>
<!--@elvariable id="self" type="com.mycompany.SubscriptionForm
"--%>
<!--@elvariable id="subscriptionForm"
type="com.mycompany.SubscriptionForm "--%>
<!--@elvariable id="cmpage"
type="com.coremedia.blueprint.common.contentbeans.Page"--%>
<cm:link target="${cmpage.linkable}" var="redirectUri"/>
<cm:link target="${self}" var="subscriptionUri">
  <cm:param name="return" value="${redirectUri}"/>
</cm:link>
<form:form id="subscriptionForm" modelAttribute="subscriptionForm"

      action="${subscriptionUri}" method="post">
  <form:input path="email"/>
  <input type="submit"/>
</form:form>
```

2. Add a handler that is able to process the subscription as well as a link scheme that builds links pointing to the handler.

```
@Link
@RequestMapping
public class SubscriptionHandler {

    @RequestMapping(value="/subscribe", method=RequestMethod.POST)
    public ModelAndView
    handleSubscription(@RequestParam(value="return", required=true)
    String redirectUri,

    @ModelAttribute("subscriptionForm") SubscriptionForm form,
    HttpServletRequest request, HttpServletResponse response)
    throws IOException {
        doSubscribe(request.getSession(), form.getEmail());
        response.sendRedirect(redirectUri);
        return null;
    }

    @Link(type=SubscriptionForm.class, parameter="return",
    uri="/subscribe")
    public UriComponents createSubscriptionLink(UriComponentsBuilder
    uri, Map<String,Object> parameters) {
        return uri.queryParam("return", (String)
        parameters.get("return")).build();
    }
    ...
}
```

Don't forget to register this class as a bean in the Spring application context.

3. Define an action substitution.

```
public class SubscriptionHandler {
    ...
    @Substitution("com.coremedia.subscription",
modelAttribute="subscriptionForm")
    public SubscriptionForm createSubscriptionSubstitution(CMAAction
original, HttpServletRequest request) {
        return new SubscriptionForm();
    }
    ...
}
```

Notes

- ➔ The parameters `original` as well as `request` are optional and might be omitted here. But in a more proper implementation it might be useful to have access to the original bean and the current request.
- ➔ The optional `modelAttribute` causes the substitution to be become available as a request attribute `subscriptionForm`. This is useful when using dealing with the Spring form tag library (see above).

4. Create a newsletter action content

- ➔ Create a content of type `CMAAction`
- ➔ Set the `id` property to value `com.coremedia.subscription`
- ➔ Insert this content to a page's teaser link list.

Here is what happens when opening the page by sending an HTTP request:

1. The request will be accepted by the `PageHandler` that builds a `ModelAndView` containing the `Page` model. This model's tree of content beans contains the new `CMAAction` instance.
2. The model will be rendered by initially invoking `Page.jsp` for the `Page` bean.
3. When the `CMAAction` is going to be rendered in the teaser list, the template `CMAAction.asTeaser.jsp` is invoked. This template substitutes the `CMAAction` bean by invoking the `cm:substitute` function while using the ID `com.coremedia.subscription`.
4. The substitution framework invokes the method `#createSubscriptionSubstitution` after checking whether `SubstitutionRegistry#register` has been invoked by any handler for his ID (which hasn't happened here). As the result, the substitutions result is a bean of type `SubscriptionForm`.
5. The above mentioned template `CMAAction.asTeaser.jsp` therefore delegates to `SubscriptionForm.asTeaser.jsp` then.

6. While rendering `SubscriptionForm.asTeaser.jsp`, a link pointing to this form bean is going to be built. The method `#createSubscriptionLink` is chosen as a link scheme so that the link points to the handler method `#handleSubscription`.
7. After the user has received the rendered page, he might enter his email address and press the submit button.
8. This new (POST) request is accepted by the mentioned handler method `#handleSubscription` that performs the subscription and redirects the original page then so that the first step of this flow is repeated.

Of course, a more proper implementation could mark the subscription state (subscribed or not) in a session/cookie and would return an `UnsubscribeForm` from `#createSubscriptionSubstitution` depending on this state.

Webflow Actions

Spring Webflow (<http://www.springsource.org/spring-web-flow>) is a framework for building complex form based applications consisting of multiple steps. Webflow based actions can be integrated into *Blueprint* as well. This section describes the steps of how to integrate this kind of actions.

In *CoreMedia Blueprint* the `PageActionHandler` takes care of generally handling Webflow actions. The flow's out coming model is automatically wrapped into a bean `WebflowActionState`. A special aspect of this bean is that it implements `HasCustomType` and therefore is able to control the lookup of the of the matching template.

1. Place your flow definition file somewhere below a package named `webflow` somewhere in the classpath. The name of the flow definition file should be `<action_id>.xml`. Example: For an action `com.mycompany.MyFlowAction` you might create a file `com.mycompany.MyFlowAction.xml` that can be placed below a package `com.coremedia.blueprint.mycompany.webflow`.
2. For every flow view (such as "success" or "failure") create a JSP template. The template name needs to match the action id. Example: The action `com.mycompany.MyFlowAction` requires templates to be named `.../templates/com.mycompany/MyFlowAction.<flowView>.jsp`. These templates will be invoked for the mentioned beans of type `WebflowActionState`.
3. Create (and integrate) a new document of type `CMAAction` and set the property `id` to the action id (such as `com.mycompany.MyFlowAction`) and the property `type` to `webflow`.

6.3.14 Images

Requirements

For a website images are required in different sizes and formats. For example, teaser need a small image with an aspect ratio of 1:1 in the sidebar and an aspect ratio of 4:3 in the main section. Images in articles and galleries are shown in 5:2 or 4:3 with a large size. And even these sizes are different on mobile devices and desktop displays.

Solution

CoreMedia Blueprint supports different formats combined with different sizes. It comes with nine predefined cropping definitions, which are used on the Perfect Chef and Aurora example sites.

- `portrait_ratio20x31` (aspect ratio of 2:3.1)
- `portrait_ratio3x4` (aspect ratio of 3:4)
- `portrait_ratio1x1` (aspect ratio of 1:1)
- `landscape_ratio4x3` (aspect ratio of 4:3)
- `landscape_ratio16x9` (aspect ratio of 16:9)
- `landscape_ratio2x1` (aspect ratio of 2:1)
- `landscape_ratio5x2` (aspect ratio of 5:2)
- `landscape_ratio8x3` (aspect ratio of 8:3)
- `landscape_ratio4x1` (aspect ratio of 4:1)

A list of sizes can be defined for each format in the `Responsive Image Settings`, located in `Options/Settings/CMChannel`. The website will automatically choose the best matching image depending of the viewport of the client's browser.

How to configure image sizes

The struct `responsiveImageSettings` contains a list of string properties. This string must contain the name of a cropping format. For example `portrait_ratio1x1`. Each format contains a list of string properties, representing one size of this format. The name and the order of this list is not important and will be ignored. Every size must contain two integer properties `width` and `height`.

If site specific image variants are enabled, the Responsive Image Settings will be used for the image editor as well. In this case the additional integer property fields `widthRatio`, `heightRatio`, `minWidth` and `minHeight` must be defined. Additionally, the field `previewWidth` and/or `previewHeight` should be defined to define the preview size in the Studio.

For example a Responsive Image Settings with two formats. `portrait_ratio1x1` with just one size and `landscape_ratio4x3` with 3 sizes.

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <StructProperty Name="responsiveImageSettings">
    <Struct>
      <StructProperty Name="portrait_ratio1x1">
        <IntProperty Name="widthRatio">1</IntProperty>
        <IntProperty Name="heightRatio">1</IntProperty>
        <IntProperty Name="minWidth">200</IntProperty>
        <IntProperty Name="minHeight">200</IntProperty>
        <IntProperty Name="previewWidth">400</IntProperty>
      </Struct>
      <StructProperty Name="0">
        <Struct>
          <IntProperty Name="width">60</IntProperty>
          <IntProperty Name="height">60</IntProperty>
        </Struct>
      </StructProperty>
    </Struct>
    <StructProperty Name="landscape_ratio4x3">
      <IntProperty Name="widthRatio">4</IntProperty>
      <IntProperty Name="heightRatio">3</IntProperty>
      <IntProperty Name="minWidth">1180</IntProperty>
      <IntProperty Name="minHeight">885</IntProperty>
      <IntProperty Name="previewWidth">400</IntProperty>
    </Struct>
    <StructProperty Name="0">
      <Struct>
        <IntProperty Name="width">200</IntProperty>
        <IntProperty Name="height">150</IntProperty>
      </Struct>
    </StructProperty>
    <StructProperty Name="1">
      <Struct>
        <IntProperty Name="width">320</IntProperty>
        <IntProperty Name="height">240</IntProperty>
      </Struct>
    </StructProperty>
    <StructProperty Name="2">
      <Struct>
        <IntProperty Name="width">640</IntProperty>
        <IntProperty Name="height">480</IntProperty>
      </Struct>
    </StructProperty>
  </Struct>
</Struct>
```

Every image cropping format must contain one image size, otherwise the default size and format, defined in `ImageFunctions`, will be used.



High Resolution/Retina Images

CoreMedia Blueprint supports high resolution images. Set the BooleanProperty `enableRetinaImages` to true. If enabled, the Javascript `jquery.coremedia.responsiveimages.js` is choosing a larger image according to the `devicePixelRatio` of the browser.

For Example the website wants to render an image with an aspect ratio of 4:3 and the best responsive image size is 400px : 300px. With a `devicePixelRatio` of 2, the JavaScript `jquery.coremedia.responsiveimages.js` is now choosing the size of 800px : 600px.

Default JPEG Compression Quality

The default JPEG compression quality is 80% in *CoreMedia Blueprint*. This parameter is configured in `blueprint-handlers.xml` for the `transformedBlobHandler`. For further information consult the "CAE Application Developer Manual", chapter "Image Transformation API".

6.3.15 URLs

Link generation and request handling is based on the concepts of the CAE web application. For further information consult the "CAE Application Developer Manual". *CoreMedia Blueprint* offers a simple mechanism for link building and parsing that is based on regular expressions. The out of the box configuration has been made with "SEO Search Engine Optimization" in mind:

- URLs show to which site section the currently displayed page belongs
- URLs for asset detailed pages – opposed to section overview pages – contain the title of the asset

See [Section 9.7, "Link Format" \[478\]](#) for link schemes and controllers of *CoreMedia Blueprint* as well as existing post processors.

6.3.16 Vanity URLs

Requirements

Editors should be able to define special URLs to special content objects which are easy to remember.

Solution

Vanity URLs are special human readable URLs which do not contain any technical identifiers like document IDs. *CoreMedia Blueprint* provides a means to assign vanity URLs to content objects.

Vanity URLs are configured in channel settings. Typically, there is one Vanity URL settings document for the root channel of a given site. This is the setup chosen for *CoreMedia Blueprint* demo content. To find the Vanity URL settings document, open the root channel of a site and switch to the `Settings` tab. You will find the Vanity URL settings document link inside the `Linked Settings` section.

Vanity URLs are defined as a relative URI path. The path might consist of several segments, but if you would like keep your Vanity URLs simple, just use only one path segment. The URI path is then prepended with a path segment consisting of the site name. For example, for the site `perfectchef`, a URI path of `my/special/artilce` would yield the Vanity URL `/perfectchef/my/special/artilce`.

To add a Vanity URL for a document, follow these steps:

1. Select the `StructListProperty vanityUrlDefinition` and create a new child Element Struct by clicking the **[Add item to List Property]** symbol in the toolbar.
2. Create a new `LinkProperty` and name it "target".
3. Set the content type field to the type of your target document.
4. Click on the value field, this will open the library window. Drag your target document from the library window into the value field.
5. Create a `StringProperty`, name it `id` and type your vanity URI path inside the value field.

Once the settings document is published, the new Vanity URL is reachable on the live site, and it is used for all generated links referring to the target document.

6.3.17 Content Visibility

Requirements

Content should become available online only within a specific time frame. For example, editors need to ensure that a press release only becomes public at a certain day and time or an article should expire after a specific day. In addition, editors want to preview their reproduced content in the context of the website as if it was already available.

Solution

CoreMedia Blueprint supports restricting the visibility of content items by setting the optional `validFrom` and `validTo` date properties of content of type `CMLinkable`.

Table 6.17. Properties for Visibility Restriction

validFrom	
UI-Name	Valid From
Description	Content where the "valid from" date has not been reached yet is not displayed on the site yet.
validTo	
UI-Name	Valid To
Description	Content where the "valid to" date has passed is not displayed on the site anymore. By not specifying either of <code>validTo</code> or <code>validFrom</code> , an open interval can be specified to define just a start or end date.

Content is filtered in the CAE during the following two stages of request processing:

- The controller is resolving content from a requested URL. See `ContentValidityInterceptor`.
- In the content bean layer whenever references to other content beans that implement `ValidityPeriod` are returned.

In the CAE visibility checking is implemented as part of an extensible content validator concept. The generic `ValidationService` is configured with a `ValidityPeriodValidator` to filter content when it is requested.

To allow editors to preview content for a certain preview date and time a `PreviewDateSelector` component has been added to *Studio*, which sets the request parameter `previewDate`. This parameter is respected by the `ValidityPeriodValidator`.

6.3.18 Content Type Sitemap

Configuration

The content type Sitemap has three fields you can configure:

The screenshot shows a web application interface for configuring a 'Sitemap' content type. At the top, there's a blue header bar with 'English (Unit...)' on the left and 'Sitemap' with a checkmark icon on the right. Below the header, there are two tabs: 'Content' and 'System', with 'Content' being the active tab. The main area contains several form fields: 'Sitemap Title' with a text input field containing 'Sitemap'; 'Root Page' with a dropdown menu showing 'Perfect Chef USA' and a globe icon; 'Teaser Title' with a text input field containing 'Enter a teaser title here.'; and 'Sitemap Depth' with a text input field containing 'Enter the depth of the sitemap here.'.

Figure 6.18. Content Type Sitemap

Enter a Sitemap Title which will be rendered as the headline of the Sitemap section in the site. The Root Page field defines the root node from where the content for the sitemap will be rendered. Additionally, the Sitemap can be rendered to a specific depth which can be set here. This depth is three by default.

6.3.19 Robots File

Requirements

Technical editors should be able to adjust site behavior regarding robots (also known as crawlers or spiders) from search engines like Google. For example:

- ➔ Enable/disable crawling of certain pages including their sub pages.
- ➔ Enable/disable crawling of certain single documents.
- ➔ Specify certain bots to crawl different sections of the site.

To support this functionality most robots follow the rules of `robots.txt` files like explained here: <http://www.robotstxt.org/> For example, the site "Corporate" is accessible as `http://corporate.blueprint.coremedia.com`. For all content of this site the robots will look for a file called `robots.txt` by performing an HTTP GET request to `http://corporate.blueprint.coremedia.com/robots.txt` A sample `robots.txt` file may look like this:

```
User-agent: Googlebot,Bingbot
Disallow: /folder1/
```

Example 6.2. A `robots.txt` file

```
Allow: /folder1/myfile.html
```

Solution

Blueprint's cae-base-lib module provides a RobotsHandler which is responsible for generating a robots.txt file. A RobotsHandler instance is configured in blueprint-handler.xml. It handles URLs like http://corporate.blueprint7.coremedia.com:49080/blueprint/servlet/robots/corporate

This is a typical preview URL. In order to have the correct external URL for the robots one needs to use Apache rewrite URLs that forwards incoming GET requests for http://corporate.blueprint7.coremedia.com/robots.txt to http://corporate.blueprint7.coremedia.com:49080/blueprint/servlet/robots/corporate

The RobotsHandler will be responsible for requests like this due to the path element /robots The last path element of this URL (in this example /corporate will be evaluated by RobotsHandler to determine the root page that has been requested. In this example "corporate" is the URL segment of the Corporate Root Page. Thus, RobotsHandler will use Corporate root page's settings to check for Robots.txt configuration.

To add configuration for a Robots.txt file the corresponding root page (here: "Corporate") needs a setting called Robots.txt

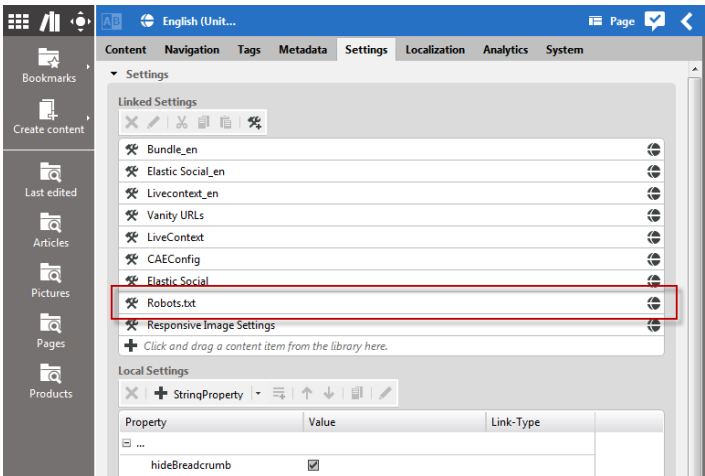


Figure 6.19. Robots.txt settings

Example configuration for a Robots.txt file

The settings document itself is organized as a StructList property like in this example:

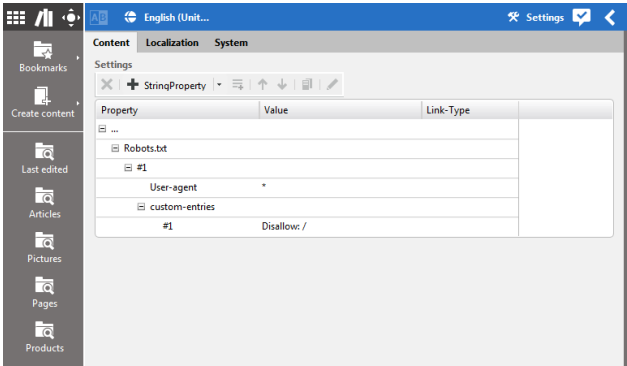


Figure 6.20. Channel settings with configuration for Robots.txt as a linked setting on a root page

For any specified user agent the following properties are supported:

- User-agent: Specifies the user agent(s) that are valid for this node.
- Disallow: A link list of items to be disallowed for robots. This list specifies a black list for navigation elements or content: Elements that should not be crawled. Navigation elements will be interpreted by "do not crawl elements below this navigation path". This leads to two entries in the resulting robots.txt file: one for the link to the navigation element and one for the same link with a trailing '/'. The latter informs the crawler to treat this link as path (thus the crawler will not work on any elements below this path). Single content elements will be interpreted as "do not crawl this document"
- Allow: A link list of items to be explicitly allowed for robots. This list specifies navigation elements or content that should be crawled. It is interpreted as a white list. Usually one would only use a black list. However, if you intend to hide a certain navigation path for robots but you want one single document below this navigation to be crawled you would add the navigation path to the disallow list and the single document to the allow list.
- custom-entries: This is a String List to specify custom entries in the Robots.txt. All elements here will be added as a new line in the Robots.txt for this node.

The example settings document will result in the following robots.txt file:

```
User-agent: *
Disallow: /corporate/corporate-information/
Allow: /corporate/corporate-information/contact-us

User-agent: Googlebot
Disallow: /corporate/embedding-test
```

Example 6.3. robots.txt file generated by the example settings

6.3.20 Sitemap

Requirements

If you run a public website, you want to get listed by search engines and therefore give web crawlers hints about the pages they should crawl. <http://www.sitemaps.org/> declares an XML format for such sitemaps which is supported by many search engines, especially from Google, Yahoo! and Microsoft.

"Sitemap" in terms of <http://www.sitemaps.org/> is not to be mistaken with a human readable sitemap which visualizes the structure of a website (see [Section 6.3.18, "Content Type Sitemap" \[302\]](#)). It is rather a complete index of all pages of a site. A simple sitemap file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9
    http://www.sitemaps.org/schemas/sitemap/0.9/sitemap.xsd">
  <url>
    <loc>
      http://helios.coremedia.com/perfectchef/spicy-duck-694
    </loc>
  </url>
  <url>
    <loc>
      http://helios.coremedia.com/perfectchef/share-your-recipes-696
    </loc>
  </url>
  ...
</urlset>
```

Example 6.4. A sitemap file

The size of a sitemap is limited to 50,000 URLs. Larger sites must be split into several sitemap files and a sitemap index file which aggregates the sitemap files. A sitemap index file looks like this:

Maximum number of URLs

```
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <sitemap>
    <loc>http://helios.coremedia.com/sitemap1.xml.gz</loc>
    <lastmod>2014-03-31T15:33:26+02:00</lastmod>
  </sitemap>
  ...
</sitemapindex>
```

Example 6.5. A sitemap index file

Solution

A sitemap consists of multiple entities (the index and the sitemap files) and has dependencies on almost the whole repository. If a new content is created, which "coincidentally" occurs in the first sitemap file, the entries of all subsequent sitemap files are shifted. In border cases even the number of sitemap files may change, which affects the sitemap index file. So you cannot generate single sitemap entities on crawler demand, asynchronously and independent of each other, but you must generate a complete sitemap which represents a snapshot of the repository. Moreover, the exhaustive dependencies make sitemaps practically uncacheable, and the generation is expensive. For these reasons *Blueprint* does not render sitemaps on demand but pregenerates them periodically. So you must distinguish between sitemap generation and sitemap service. Both are handled by the live web application, though.

Sitemap Generation

CoreMedia Blueprint features separated sitemaps for each site. Sitemap generation depends on some site specific configuration, like the document types to include or paths to exclude, amongst others. This configuration is specified by `Sitemap Setup` Spring beans. The `lc` and the `corporate` extension each provide a `SitemapSetup` bean suitable for their particular sites. Projects can declare their own sitemap setups. The setups are collected in the `sitemapConfigurations` Spring map.

```
<bean id="livecontextSitemapConfiguration" class="c.c.b.c.s.SitemapSetup">
  <property name="protocol" value="http"/>
  ...
</bean>

<customize:append id="appendLSC" bean="sitemapConfigurations">
  <map>
    <entry key="livecontext" value-ref="livecontextSitemapConfiguration"/>
  </map>
</customize:append>
```

If you want to generate a sitemap for a site, you have to specify the setting `sitemapOrgConfiguration` at the root channel. It is a `String` setting, and the value must be a key of the `sitemapConfigurations` map.

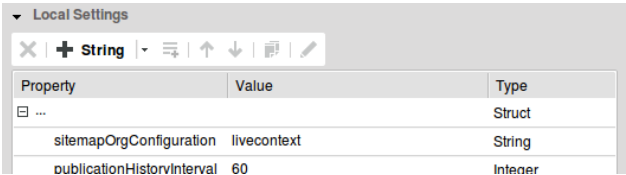


Figure 6.21. Selection of a sitemap setup

By default, the PerfectChef sites and the Corporate sites are sitemap-enabled, while the Aurora sites are not. Since the Aurora sites serve only as backend for WCS applications, there is no need for sitemaps.

Sitemaps are generated periodically in the Delivery CAE by a `SitemapGenerationJob`. You can specify the initial start time and the period as application properties `blueprint.sitemap.starttime` and `blueprint.sitemap.period`, respectively. For details about the values see the JavaDoc of the setters in `SitemapGenerationJob`. The *Blueprint* is preconfigured to run the sitemap generation nightly at 01:30. You can also trigger sitemap generation for a particular site manually by the URL

```
http://live-cae:49080/blueprint/servlet/internal/corporate-de-de/sitemap-org
```

where `corporate-de-de` stands for the segment of the site's root channel. Note that it is an internal URL which can only be invoked directly on the CAE's servlet container. Sitemap generation is an expensive administrative task, which is not to be exposed to end users. CoreMedia's default Apache rewrite rules block internal URLs, see `rewrite.inc` files.

The sitemaps are written into the file system under a directory which is specified by the `blueprint.sitemap.target.root` application property. That means, the CAE needs write permissions for this directory.

Sitemap Service

The generated sitemaps are available by the URL pattern

```
/service/sitemap/the-site-ID/sitemap_index.xml
```

In order to inform search crawlers, the sitemap URLs are included in the `robots.txt` files. Since there is only one robots file per web presence, you will see multiple sitemap entries for the localized sites:

```
User-agent: *
Disallow: /

Sitemap: http://corporate.acme.com/service/sitemap/ab..ee/sitemap_index.xml
Sitemap: http://corporate.acme.com/service/sitemap/lc..7a/sitemap_index.xml
```

6.3.21 Website Search

Requirements

For IBM WebSphere Commerce integration scenarios, all search is handled by IBM WCS. CMS content must be crawled by the IBM Solr Search engine. Please refer to the IBM documentation. A configuration file for each example site is part of the *CoreMedia LiveContext 2.0 WebSphere Commerce Project Workspace* archive (for example, `WCDE-ZIP/components/foundation/subcomponents/search/solr/home/droidConfig-cm-aurora-en-US.xml`).



In order to make content more accessible for their audience virtually all websites have full-text search capabilities. To improve the search experience some websites

also offer features such as search term autocompletion, suggestions in case of misspelled search terms, more advanced filtering options or even metadata based drilldown navigation in search results.

Solution

CoreMedia CMS has built-in integration with the Apache Solr search engine. *Blueprint* comes with a small abstraction layer that offers unified search access to Solr for all CAE based code. It provides the following features, all based on standard Solr functionality:

- Full text search: Search for content across all fields
- Field based filters: Filter results by metadata such as the content type, the site section it belongs to, etc.
- Facets: Display facets, that is the number of results in a field for certain values
- Spellcheck suggestion: "Did you mean" suggestions for misspelled terms
- Search term highlighting: All words are highlighted in your text
- Validity range filtering: Automatically filter for only visible results (see section [Section 6.3.17, "Content Visibility" \[301\]](#))
- Filter non-searchable: Automatically filter content that should not be part of search results.
- Caching: Search results can be optionally cached for a certain amount of time.

The search integration can be found in the modules `com.coremedia.blueprint.cae.search` and `com.coremedia.blueprint.cae.search.solr`.

6.3.22 Search Landing Pages

Feature is only supported in *e-Commerce Blueprint*



Requirements

Using *CoreMedia Digital Experience Platform 8* the user should have the possibility to define custom page layouts for search terms.

Solution

Search Landing Pages are used to apply a custom page layout for product searches that match specific search terms. This feature is used when CAE fragments should

be included to search result pages of a commerce system. To provide a new search landing page, do the following:

1. Create a new folder with the name `Search Landing Pages` in one of your sites folders. The folder must be part of a site, global search landing pages are not provided.
2. Create a new page document and add the matching keywords in the input field "HTML Keywords" (CMChannel property "keywords").
3. Add the newly created page document as navigation child to the root document. Ensure that the search landing page has checked the "Hidden in Sitemap" and "Hidden in Navigation" checkboxes.

When the search landing page is included to the commerce storefront, only the main placement of the page's page grid will be included as fragment.

6.4 Website Development with Themes

A consistent page design is essential for a professional website. Apart from the HTML structure reflected by the templates, the layout is mainly controlled by web resources, like CSS, JavaScript and templates. CoreMedia uses themes to bundle these files.

Consistent page design with themes

Developing and using themes, have some conflicting interests. On the one hand, changes of web resources should be immediately effective on your site, so they must be integrated into the caching and invalidation mechanisms of CoreMedia CMS and thus be maintained in the content repository. On the other hand, web designers want to work with their favorite familiar tools and short round-trips to test their changes. Maintaining each interim change into the content repository would be too much effort. In many projects the CSS and JavaScript is maintained by external agencies which do not even have access to the CMS but deliver their work as ordinary files.

Conflicting interests between developing themes and using themes

In order to resolve this conflict, Web resources are treated as content in *Blueprint*, so that you do not need to take care of dependencies such as making sure that image files linked to the CSS files are deployed into your web application and how the files are included to your site. On the other hand, *CoreMedia Blueprint* ships with built in, ready to use features for developers, who would like to get started working on local web resources and templates as themes for their site.

Develop locally but have resources as content

One has to differentiate between the pure theme concept which comprises the theme structure, the theme descriptor file and the *coremedia-webresource-content-maven-plugin* and the automated Blueprint build process.

Theme concept and Blueprint build process

The following sections describe these topics:

- [Section 6.4.1, “CoreMedia Themes” \[311\]](#) describes the structure of a theme and gives some hints about coding styles.
- [Section 6.4.2, “Web Development Workflow” \[321\]](#) describes the web development workflow. That includes how developers can work with local resources, rather than with content objects inside the repository, how deployable theme artifacts can be build and how the artifacts can be imported into the content repository.

6.4.1 CoreMedia Themes

In order to edit the web resources you need to know how the workspace is arranged. Web resources are organized in themes, where each theme is a separate module. A theme consists of a set of related resources, typically

- Templates (FreeMarker)
- CSS

- ➔ JavaScript
- ➔ Images
- ➔ Fonts
- ➔ Third-party libraries (for example, jQuery)

A site can aggregate multiple themes, so you can easily share common web resources among different sites. *CoreMedia Blueprint* currently contains the following themes for the example websites. Some extensions, such as asset management, contain additional web resources:

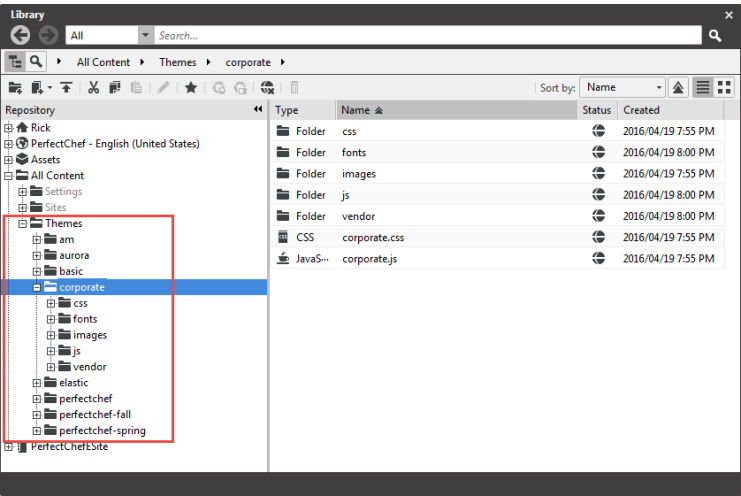


Figure 6.22. Themes in the Library

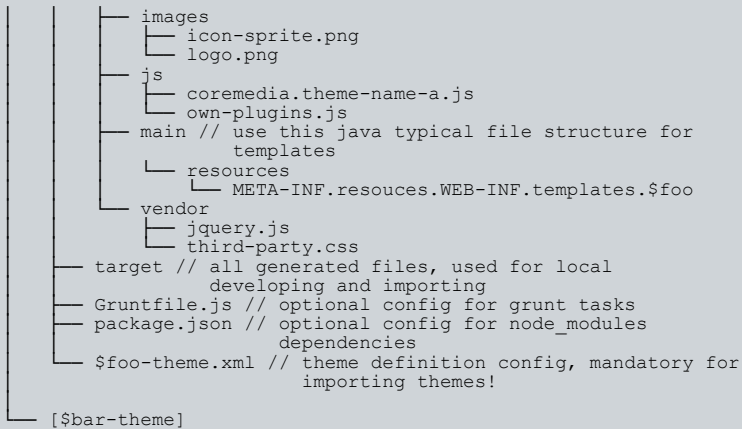
Theme Structure

A theme consists of the actual web resources and the theme descriptor. [Example 6.6, “File structure of a theme” \[312\]](#) shows the directory structure of a theme in the workspace. CoreMedia recommends to use a single root directory for each theme, in order to avoid conflicts between multiple themes.

Workspace structure

```
themes
├── [foo-theme] // name of theme,
│               for example "foo" and suffix "-theme"
│   └── src // all source files. add subfolders for all types like
│       │
│       │       sass, js, fonts or images
│       │       ├── css
│       │       │   ├── box.css
│       │       │   └── navigation.css
│       │       ├── fonts
│       │       │   └── arial.woff
```

Example 6.6. File structure of a theme



Below the root directory of the theme module lies the theme descriptor. It is named after the theme module to which it belongs, `basic-theme.xml`, for instance. The theme descriptor contains the paths to all Javascript and CSS files used by the theme. This file is used to create the aggregating CSS and JavaScript files in the CoreMedia content. Here, you have to add paths to CSS and JavaScript files of other themes that you want to use in your theme. [Example 6.7, “Theme descriptor example” \[313\]](#) shows the structure of a theme descriptor and contains some comments for the usage.

Theme descriptor

```

<?xml version="1.0" encoding="UTF-8"?>
<themeDefinition modelVersion="1">
  <!--
    name *mandatory: - used as name for the aggregating files
    like theme-name.js or theme-name.css
  -->

  <name>theme-name</name>

  <!-- description: optional -->

  <description>Basic Theme</description>

  <!--
    add all javascript and css files needed for this theme. the
    paths are relative to the content object. Don't use
    absolute or workspace path. Take care about the
    order of the files, if they are related to each
    other, like jquery and jquery plugins.
  -->

  <!--
    all files are linked to a placeholder js file called
    theme-name.js
  -->
  <javaScripts>

```

Example 6.7. Theme descriptor example

```

<!--
  "disableCompression" optional attribute that sets the
  "disableCompress"
  flag of the resulting content object. Set this flag if the
  file is already
  compressed or if it is not compatible with the compression
  engine used
  by the CAE.
-->
<javaScript
disableCompression="true">vendor/jquery-2.2.3.min.js</javaScript>

<javaScript>js/example.js</javaScript>
<javaScript>../other-theme/js/example.js</javaScript>

<!--
  "ieExpression" optional expression for conditional comments
  for Internet Explorer without "if", see
  https://msdn.microsoft.com/en-us/library/ms537512.aspx#syntax
-->
<javaScript ieExpression="IE">/js/all_ie.js</javaScript>
<javaScript ieExpression="lte IE 9">/js/ie9.js</javaScript>

</javaScripts>

<!--
  all files are linked to a placeholder css file called
  theme-name.css
-->

<styleSheets>
<css>css/example.css</css>
<css>../other-theme/css/example.css</css>
<css ieExpression="IE">.css/ie.css</css>
</styleSheets>

</themeDefinition>

```

All web resource files, except the templates, can be arranged arbitrarily in directories. However, In CoreMedia themes these resources are arranged by their particular types. When you import a theme into the CoreMedia repository, these directories will be mapped 1:1 to repository paths. See [Section 6.3.12, “Client Code Delivery” \[291\]](#) for more details. CoreMedia uses the following typical style for web-safe file names:

- ➔ File names should all be lower case
- ➔ Nouns should be used in singular
- ➔ Words should be separated by dashes

Structure of web resources

Templates are located in the `src/main/resources` directory of the theme module. Inside this directory, according to the Servlet spec 3.0, the templates are located in the `META-INF/resources` directory. Underneath this convention driven base path the templates are structured in packages, corresponding to the content beans. See [Section 6.3.10, “Dynamic Templating” \[288\]](#) for details.

Templates structure

In the CoreMedia repository, templates are stored as JAR archives in blob properties.

Themes imported into the *Content Server* are stored in a folder named `Themes/<ThemeName>` by default. The content is stored in the following content types:

Themes in the CoreMedia repository

- CSS files in content of type `CSS`
- JavaScript files in content of type `JavaScript`
- Freemarker Templates as JAR archives in blob properties in content of type `Template Set`
- All other resources in content of type `Technical Image`

In order to connect the content with a Site page, the content is linked in the following way:

- All imported CSS files of the theme are linked by a content of type `CSS` in the main folder of your theme called `<themenam>.css`
- All imported JavaScript files of the theme are linked by a content of type `JavaScript` in the main folder of your theme called `<themenam>.js`
- All images or fonts are linked from inside the CSS files, through direct content links to the corresponding `Technical Image` content.
- Templates are found through its view repositories. See [Section 6.3.10, “Dynamic Templating” \[288\]](#) for details.
- The Site page links to the aggregating `CSS` and `JavaScript` content items (see [Figure 6.25, “Linking a theme to site root” \[330\]](#)).

CSS Files

CoreMedia HTML in templates follow the B.E.M. pattern, standing for block, element, modifier, for naming. Make sure you understand the principles (see for example <http://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax>).

Usage of IDs

Avoid styling IDs. IDs should be used for the semantic of a page. CSS rules bound to IDs have a higher order than CSS rules bound to class names. This leads, if not used carefully, in long term to very bad workarounds in CSS styles and so to much more CSS styles than needed and to very difficult maintenance.

In general only elements that occur exactly once inside a page are valid candidates for IDs. However, this doesn't mean that it is a good idea to make them an ID. Only use IDs in these two cases:

- You need to identify a DOM element where you want to explicitly say that it may only occur once per page and you need to make sure that this is the case. This is useful to jump to specific sections of a page, for a screen reader, for instance.
- You need to distinct an element of a certain common class from the other ones and you can not find it through relatively moving over the DOM tree.

When you are in doubt, use a class name.

CoreMedia CSS files follow a style guide to which you should adhere when you write your own stylesheets.

Coding Style

- In class names use dashes to separate, not camelCase or underscores.
- Indent property declarations by two spaces.
- In property declarations, put a space after the colon.
- In rule declarations, put spaces before the curly bracket.
- In parameter listings, put a space after each comma.
- Write empty property values without a unit. That is "width: 0" instead of "width: 0px".
- Use hexadecimal color codes (#00000a) with lowercase instead of RGB unless you are using RGB.
- Use 6-character hexadecimal code, if you use 3-character code every character is duplicated. That is, for instance, #abc is short for #aabbcc and not #abcbcc.

```
.styleguide-format {
  margin: 0;
  border: 1px solid #0f0;
  color: #000000;
  background: rgba(0, 0, 0, 0.5);
}
```

Example 6.8. CSS code that follows the style guide

Saas Files

In the *Brand Blueprint* CSS files are generated from Saas files (see sass-lang.com). Except for the main Saas file of the theme (`$theme-name.scss`) all other files are partial files. That is, the name starts with an underscore so that Saas does not render separate files. In each folder is a `_import.scss` file in which all the other

partial files from the folder are imported. This is required by Idea, to make all files "green". For Saas it is enough to link to all imported files from the main Saas file.

The folder structure is as follows:

```
sass // scss files should be located inside
      a themes 'src' folder
├── base // contains configuration in form of variables
│   ├── _import.scss
│   ├── _variables.scss
│   └── ...
├── components // contains components with declarations
│   ├── _component1.scss
│   ├── _component2.scss
│   └── ...
├── utils // contains mixins and functions to be used by
│   │   components
│   ├── _import.scss
│   ├── _utils1.scss
│   ├── _utils2.scss
│   └── ...
└── $theme-name.scss // main Sass file for theme. Imports
    all required Sass files
```

Example 6.9. Folder structure of the Saas files

Commenting in Saas

The following conventions for comments are used:

- ➔ Comments for SASS are made with `/**/`. They do not appear in the generated CSS and can be used for internal comments.
- ➔ Comments appearing in the generated CSS are made with `/* */`. These comments are removed when minifying the CSS in the CAE.
- ➔ Comments appearing in the generated CSS `/*! */`. These comments are not removed when minifying the CSS in the CAE. Used for license texts, for example.

JavaScript Files

In the CoreMedia frontend, JQuery is used as the main JavaScript framework. You should also use this framework for your own extensions.

Using a selector in JQuery is an expensive operation and should only be done once. If you want to use the selector multiple times, store it in a variable. In the example, the variable starts with a \$, so that it is clear, that it is a jquery object.

Save selector in a variable

```
Bad example

$(".container .children").addClass("black");
$(".container .children").show();

Avoids redundancy but still bad example
```

Example 6.10. Save selector in variable

```
var selector = ".container .children";
$(selector).addClass("black");
$(selector).show();

Good example

var $children = $(".container .children");
$children.addClass("black");
$children.show();
```

In order to avoid conflicts with different JavaScript frameworks the `noConflict(true)` functionality of jQuery is used to reset the assignment of `$` and also of `jQuery` global scope variables. While `$` avoids conflicts with other JavaScript frameworks such as Dojo, removing jQuery assignment also makes sure that there are no conflicts with different jQuery versions used.

jQuery and noConflict

For CoreMedia JavaScript this has the consequence that you cannot rely on `$` or `jQuery` as a variable of the global scope. The jQuery functionality used in *CoreMedia Blueprint* is attached to the variable `coremedia.blueprint.$`

This can be realized without much refactoring to the JavaScript Code if you declare a local variable `$` which is only valid in the current scope (`var $ = coremedia.blueprint.$`). CoreMedia jQuery plugins already have been adjusted in a more elegant way, where the correct jQuery Version is injected into the function registering the plugin (see for example, `jquery.responsiveImages.js`).

Images

For images exist no specific rules. Images are imported in `Technical Image` content items. In your CSS or JavaScript files in the workspace, you link to images through a relative path URL. For example, `background-image: url("../images/testimage.png")`. After the upload, these links are replaced by internal content links.

Templates

Dynamic templating (see [Section 6.3.10, “Dynamic Templating” \[288\]](#)) requires the usage of Freemarker, not JSP, templates. Freemarker templates are imported as JAR files into a blob property of content of type `Template Set`. See CoreMedia Content Application Developer Manual for more details about templates.

Templates naming and lookup

The view dispatcher of the CAE (see the CoreMedia Content Application Developer Manual for more details) selects the appropriate view template for a content bean according to the following data:

1. Name of the content bean

The view dispatcher looks for a template whose name starts with the name of the content bean.

Example: The template `CMExample.ftl` is a detail view for the content bean `CMExample`.

2. A specific view name

A view name specifies a special view for a content bean. The view is added as a parameter when you include a template in another template via `<cm.include self=self view="asPlacement"/>`.

Example: The template `CMExample.specialView.ftl` is a special view for the content bean `CMExample`.

3. A specific view variant

A view variant is used, when the look of a rendered view should be editable in the content (see [Section 6.3.7, "View Types" \[282\]](#) for details).

Example: The template `CMExample.[differentLayout].ftl` is a special view of the content bean `CMExample`. The view variant must be enclosed in square brackets.

The template name is always in the order content bean name, view name, view variant. The view dispatcher looks for the most specific template.

Freemarker

Escaping HTML output

In *CoreMedia Blueprint* escaping of output is enabled by default. Auto-escaped are all values printed with `${value}`. However, there are cases where you need to disable escaping, for example, when you get HTML code that you want to print as HTML, not escaped HTML.

When you really need to disable auto-escaping (not recommended) you can use the `cm.unescape` plugin.

```
<@cm.unescape self.textAsHtml />
```

Example 6.11. Disable auto-escaping with the `cm.unescape` plugin

Robustness of templates

In order to make sure that the rendering of templates does not fail you have to ensure that FTL template can be rendered although some information is not provided. In order to achieve this FreeMarker adds some functionality to detect if a variable is set and if it contains content.

If you want to check for existence and emptiness of a hash/variable (null is also considered as empty) you need to use `?has_content`.

If you want to declare a default value for an attribute that could be null or empty use `!` followed by the value to be taken if the variable/hash is null.

Example:

```
${existingPossibleNullVariable!"Does not exists"}
<#list existingPossibleNullList![] as item>...</#list>
```

Example 6.12. Example of a fallback in Freemarker

Freemarker for JSP Developers

As a JSP developer you are familiar with JSPs in general and with writing CAE templates with JSPs. In this section, you will learn about important differences.

Type-Hinting

Type-hinting in JSP or Freemarker templates helps IntelliJ Idea to offer you code completion and to make the templates "green". The syntax of the required comments is different in Freemarker than in JSP:

- ➔ Comments are marked with `<#-- comment -->` instead of `<%-- comment --%>`
- ➔ The annotation is called `@ftlvariable` instead of `@elvariable`
- ➔ The attribute that names the typ-hinted object is called `name` instead of `id`
- ➔ The comment must have a single space after the opening comment tag

```
JSP:
<%--@elvariable id="self"
type="com.coremedia.blueprint.MyClass"--%>

Freemarker:
<#-- @ftlvariable name="self"
type="com.coremedia.blueprint.MyClass" -->
```

Example 6.13. Difference between JSP and Freemarker type-hinting comment

Passing Parameters

In JSP files, it was necessary to wrap arguments passed to taglibs or other functionality into quotes and to give them out via `${}`. In Freemarker, this is no longer necessary.

```
JSP:
<mytaglib:functionality name="${name}" booleansetting=true />
```

Example 6.14. Passing parameters


```
Freemarker:
  <mymacro.functionality name=name booleansetting=true />
```

6.4.2 Web Development Workflow

This section contains the best practice web development workflow of *CoreMedia*. It describes how to adapt your resource files in the CoreMedia workspace with fast turnaround times and how you can deploy the files to the live system later (see [Section 6.3.12, “Client Code Delivery” \[291\]](#) for details). It does not cover how to write CSS or JavaScript files or how to configure and use the *CoreMedia CAE*.

Web development usually takes place in IDEs or some other kind of source code editor. And since development of web resources, aside from minor changes, shouldn't take place in *CoreMedia Studio*, *CoreMedia Blueprint* has a solution, that lets web developers work with resource files in the workspace until the files are ready to be imported into the content repository.

CoreMedia Blueprint supports local resources as a simple yet powerful way for developers to work with workspace resources, rather than code objects in the content repository.

Develop local, deploy global

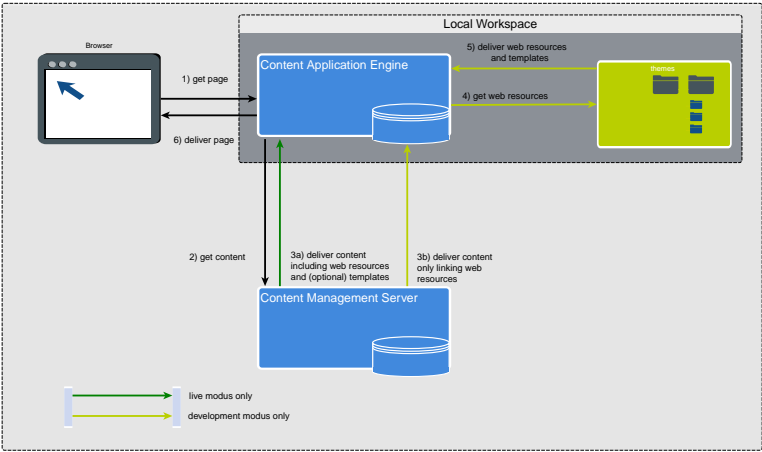


Figure 6.23. CAE flow in detail

Figure 6.23, “CAE flow in detail” [321] gives an overview of the idea behind local resources.

1. The browser requests a page from the locally started CAE.
2. The CAE requests the content from the Content Server.

3. While in development mode, the Content Server delivers content such as Articles and content items which link to the web resources.

In live mode, the Content Server also delivers the web resources to the CAE.

4. The CAE has got the editorial content which links to the web resources. Now, the CAE resolves the local location of the web resources and requests the resources from the file system.
5. The CAE reads the resources from the file system.
6. The CAE combines the content from the Content Server and the web resources from the file system and delivers the requested page to the browser.

Workflow

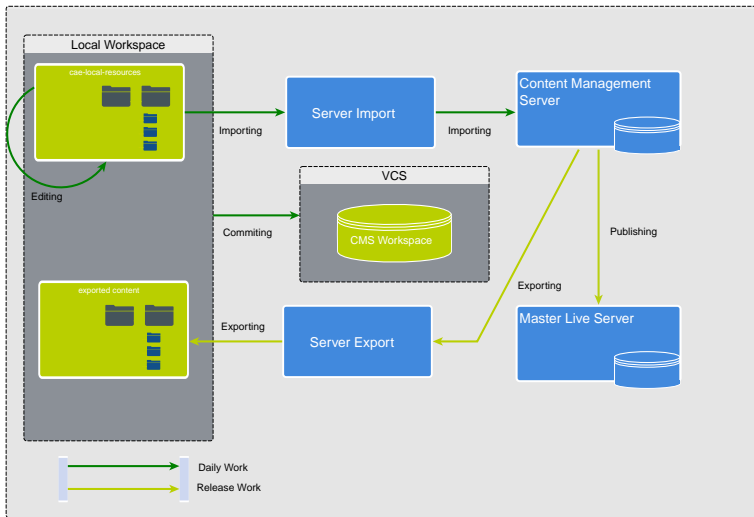


Figure 6.24. Workflow in detail

Figure 6.24, “Workflow in detail” [322] gives an overview of the web development workflow including the interaction between the different locations of web resources. The “exported content” part of the local workspace contains files used for versioning content itself for later reimport into a different *CoreMedia Content Management Server* (find more in [section “Exporting”](#) [331]).

The complete workflow comprises the following steps:

1. Editing the web resources to fit yours needs
2. Building the sources with Grunt
3. Importing the local changes to the Content Management Server and Linking the resources to the content.

This step is only necessary when you start with a new theme. Afterwards you can work with local resources until you want to publish the changes to the live system.

4. Previewing your changes in the preview CAE
5. Committing your changes (optional) to a VCS
6. Exporting the resources (optional)

Software Requirements

On your computer you need the following:

- The Blueprint workspace which contains the web resources
- Your favorite tools to develop and maintain the web resources
- A locally running preview CAE to visualize your changes immediately
- Access to *CoreMedia Studio* to attach web resources to pages

For the development of web resources it is not relevant whether the *Content Management Server* used by CAE and *Studio* is running locally or remote. So you can use shared test infrastructure.

Editing Source Files

In general, editing a theme is a straightforward development task as soon as you have set-up the preview. When you edit CSS files, SaaS files or JavaScript files, add images and, maybe, write Freemarker templates you will immediately see all changes in your preview CAE.

All CoreMedia themes come with a Grunt watch task, which includes a live reload mode. This will automatically reload your changed files (see <https://github.com/gruntjs/grunt-contrib-watch#live-reloading>).

However, before you can start editing a theme, you need a theme. You can either edit an existing theme, or create a new theme. Creating a new theme requires additional work, because before you can see the preview, you need to create a new module, do an initial upload of your theme to the *Content Server* and link it to a site.

Renaming or adding of templates will work smoothly, but deleting a template will not work without clearing the cache. Empty the cache or restart the CAE to see the affected changes.



Creating a New Theme

A theme is edited in an extension module in the workspace. The structure of a theme is described in [Section 6.4.1, “CoreMedia Themes” \[311\]](#).

When you create a new theme it is a good idea to extend at least the *CoreMedia Blueprint* basic theme because it deals with some common problems, such as conflicting JavaScript frameworks. The basic theme is located in `modules/cae/cae-themes/basic-theme`.

Extending an existing theme

Extending an existing theme requires two steps:

1. Add a dependency to this theme in the POM of your own theme. This is necessary for code completion in your IDE.
2. Add links to all CSS and JavaScript files of the theme that you extend to the theme descriptor of your new theme .

When you import resources from another theme, you need to follow a certain order:

CSS files First, you have to import the CSS files of the parent theme, then your own files.

JavaScript files

1. Vendor-specific JavaScript files of the parent theme
2. Vendor-specific JavaScript files of your theme
3. CoreMedia-specific JavaScript files of the parent theme
4. JavaScript files of your theme

The reason for this order is the use of `noConflict(true)` in the first CoreMedia-specific JavaScript. Because, CoreMedia cannot customize third-party addons, `noConflict()` can only be called after the inclusion of third-party addons. All CoreMedia-specific JavaScript files use a CoreMedia-specific namespace for JQuery to avoid conflicts. As a result, you cannot rely on `$` and **jQuery** global scope variables.

Preparing the Preview

Immediate preview of your changes requires a local preview CAE on a Tomcat in development mode and the usage of local resources.

Internally, the CAE handlers and link schemes will map the linked resource objects of a page content in the repository to the files in the local workspace. For this, you have to do the following configuration:

1. Add the paths to the resources of your theme to the `extraResourcePaths` property of the `Resources` element in the `tomcat-context.xml` file of the preview CAE Tomcat.

```
<Resources
  className="org.apache.naming.resources.VirtualDirContext
  extraResources="${project.basedir}/../../extensions/corporate/corporate-theme/src/main/resources/CAE-INF/resources,
  "${project.basedir}/../../extensions/corporate/corporate-theme/target/resources,
  "${project.basedir}/../../extensions/corporate/corporate-cae/src/main/resources
```

Example 6.15. Theme paths in `tomcat-context.xml`



All paths must be in one line, separated by commas.

- To start the local Tomcat in development mode, use the following command:

```
mvn -pl :cae-preview-webapp tomcat7:run -Pdevelopment-ports
    -Dinstallation.host=<YourCMS>
```

- The Maven setting `cae.use.local.resources` of the preview CAE must be "true" in order to use local resources. This is the default setting.
- The Maven setting `cae.developer.mode` of the preview CAE must be "true" in order to run Tomcat in developer mode. This is the default setting

Local resources only work with a CAE started in a Tomcat in development mode. Otherwise, the CAE will not initialize, throwing an `IllegalStateException`. If development mode is turned on, JavaScript and CSS content will not be minified and merged into one JavaScript and one CSS resource.

Open your browser at <http://localhost:40081/blueprint/servlet/<YourDemoSite>>.

- You have to create and link a content structure in the Content Server which corresponds to your local resource structure. The easiest way is to import your resources in the content repository as described in [Section "Import Changes into Repository and Link to Content" \[326\]](#) and link them afterwards to the site.
- In order to see the effect of your changes, you have to build your resources after each change. The easiest way is to use `grunt watch`. This will watch your Sass, Javascript and Freemarker source files and will recompile them after each change. See the `README.md` file in the `corporate-theme` module of the workspace for details.

When you have configured the preview, you will see the effect of changed web resources in the CAE in your local browser by navigating through the site that you have changed.

When you have started a local CoreMedia Studio you can watch the changes more comfortably in the Studio preview, because, by default, *CoreMedia Studio* uses the *CoreMedia CAE* for preview which is installed on the same computer as *Studio*. The Studio preview offers the ability to explicitly search for elements and display them

Preview in local Studio

as preview without displaying the surrounding sites while still loading dependencies like CSS styles from web resources.

When you do not want to build and start a local *CoreMedia Studio*, you can just copy and paste the preview URL of a non-local Studio to a new browser window/tab and change the hostname to your localhost. Therefore, you will see the preview as it would be in *Studio*.

Preview without local Studio

Committing (optional)

The next step is committing your local changes to the VCS of your choice to save and finish your daily work.

Import Changes into Repository and Link to Content

You do not need to import every change into *CoreMedia Content Server* for your daily work. Importing is only required when you create a new theme and when you add or delete web resources from your theme. That is because the linked resources in the *Content Server* have to correspond to your local resources.

The integrated *Blueprint* workflow for theme import requires the following Maven plugins:

frontend-maven-plugin

This plugin (see <https://github.com/eirslett/frontend-maven-plugin>) is used to integrate Node, NPM and tools such as Grunt, Bower, Gulp into your Maven workflow. The plugin uses Grunt to build your Saas resources and to put the Freemarker templates into a JAR file.

maven-assembly-plugin

This plugin puts all generated web resources into a Zip file and also adds the theme descriptor to the file.

coremedia-webresource-content-maven-plugin

This plugin (see <https://documentation.coremedia.com/utilities/coremedia-webresource-content-maven-plugin/> for details) creates CoreMedia XML conform content from your web resources. This content is ready for the import into the *Content Server*.

In short, the *Blueprint* workflow consists of the following steps:

1. You edit the native web resources in a theme. See [Section 6.4.1, “CoreMedia Themes” \[311\]](#) for details.
2. You build the theme with the *frontend-maven-plugin* or use directly Grunt (see the `README.md` file, in the `corporate-theme` extension).

3. The web resources are bundled into Zip artifacts with the *maven-assembly-plugin* in the theme modules.
4. The *coremedia-webresource-content-maven-plugin* takes the themes artifact and creates CoreMedia XML conform content.
5. Afterwards, you can use the *serverimport* tool (see Section “Serverimport/Server-export” in *CoreMedia Content Server Manual* for details) to import the content into the *Content Server*.
6. When you have imported the theme for the first time, you have to link it to the appropriate site. You can do this in *CoreMedia Studio*.

The following sections describe these tasks in more detail:

Building and Packaging the Web Resources

The theme is build with Grunt. Either directly started, or with the *maven-frontend-plugin*. When you want to add the templates to the theme, you have to start the plugin with the `withTemplates` profile.

Building the theme

```
grunt build
mvn install
```

Example 6.16. Building the theme with Grunt or Maven

Grunt is configured in the `Gruntfile.js` file. Mainly, Grunt processes the Saas files and copies them together with the other resources into the `target/resources/themes` folder. The Freemarker templates are packaged into a JAR file and also copied into the `target` folder.

As a second step, the *maven-assembly-plugin* takes the web resources from the `target` directory and packs them into a Zip file. It also adds the theme descriptor into the `THEME-METADATA` folder of the Zip file. The configuration of the *maven-assembly-plugin* is taken from the `assembly/resources-assembly-descriptor.xml` file.

Packaging the theme

The Maven build process also creates a theme JAR file which contains the templates, directly below the `target` folder. However, this file is not used.



Converting web resources into content items

Before you can import web resources into the CoreMedia repository, you have to convert them into an XML format that can be imported. *CoreMedia Blueprint* contains the *coremedia-webresource-content-maven-plugin* for this task (see <https://documentation.coremedia.com/utilities/coremedia-webresource-content-maven-plugin/> for details).

When configured, the plugin processes the web resources of all dependencies with type "zip" and classifier "theme" automatically during the Maven build and produces

XML content ready to be imported with the *serverimport* tool. The content files are bundled into another Zip artifact.

Configure the plugin in your POM file as follows:

```
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>MyTheme</artifactId>
    <version>${project.version}</version>
    <classifier>theme</classifier>
    <type>zip</type>
  </dependency>
</dependencies>
<build>
  <plugins>
    <!-- Create Webresource content in target folder -->
    <plugin>
      <groupId>com.coremedia.maven</groupId>

      <artifactId>coremedia-webresource-content-maven-plugin</artifactId>

      <dependencies>
        <!-- Provide the CSS importer implementation and its
configuration-->
        <dependency>
          <groupId>${project.groupId}</groupId>
          <artifactId>css-importer-lib</artifactId>
          <version>${project.version}</version>
        </dependency>
        <!-- Provide doctypes to create web resources as content-->

        <dependency>
          <groupId>${project.groupId}</groupId>
          <artifactId>contentserver-blueprint-component</artifactId>

          <version>${project.version}</version>
        </dependency>
      </dependencies>
      <executions>
        <execution>
          <id>compile-resource-corporate-testdata</id>
          <goals>
            <goal>compile</goal>
          </goals>
          <phase>compile</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Example 6.17. Configuration for webresource plugin

The module in which you execute the plugin requires a dependency on your theme artifact. The plugin itself requires a dependency on CoreMedia content types. However, the plugin is only capable of the *CoreMedia Blueprint* document types. Even though the dependency to the document types might look generic, it is needed only for technical reasons. That is because the plugin makes use of a temporary content server internally.

Dependency on content types

CoreMedia Blueprint comes with some `test-data` modules which provide the content of the CoreMedia example sites. In the default configuration, these modules include the themes content for input.

Content modules

Even though presumably you do not maintain any articles and pictures in the project workspace, you should add a content module to your project right now, because it also serves as hook point to include the themes. You might use a content module for some initial content like technical settings, symbols and the like.

Importing content into Content Server

When the *coremedia-webresource-content-maven-plugin* has created XML content, import the content into the *Content Server*. For the local Chef deployment, this is integrated in the deployment workflow. So, when you have integrated the content creation into an existing content module as described above, your theme will automatically be imported, when you create or update your Vagrant box.

Content of type `CMJavaScript` stores the JavaScript as XML in the repository. However, some JavaScript minifier tools create minified JavaScript that contains Unicode characters that are not allowed in XML. For example, `U+001F` or `U+FEFF`. If you try to load or import JavaScript with such a character, you will get error or log messages like `Could not convert markup or InvalidProperty-ValueException(errorCode: CAP-API-16176)`.



When you want to import the content manually, using the *serverimport* tool, have a look in [Section 3.5.3, “Locally Starting the Components” \[85\]](#). You will find a description on how to import content from the workspace into the server. In Section “Serverimport/Serverexport” in *CoreMedia Content Server Manual* you will find more details about the *serverimport* tool.

Linking

After importing the web resources, they are available on your *CoreMedia Content Server* below the `Themes` folder. If you added new web resources, you have to link them to the root page of a site (see [Section 6.3.12, “Client Code Delivery” \[291\]](#)). To do this, you can use *CoreMedia Studio* or *CoreMedia Site Manager*. If you just changed an existing web resource and do not want to change linking you can skip this step.

To link the theme, proceed as follows:

1. Open the Page to which you want to add the theme in *CoreMedia Studio*.
2. Open the *System* tab.
3. Link the aggregating `CSS` and `JavaScript` files from the main folder of the theme to the *Associated CSS* and *Associated JavaScripts* properties respectively (see [Figure 6.25, “Linking a theme to site root” \[330\]](#)).

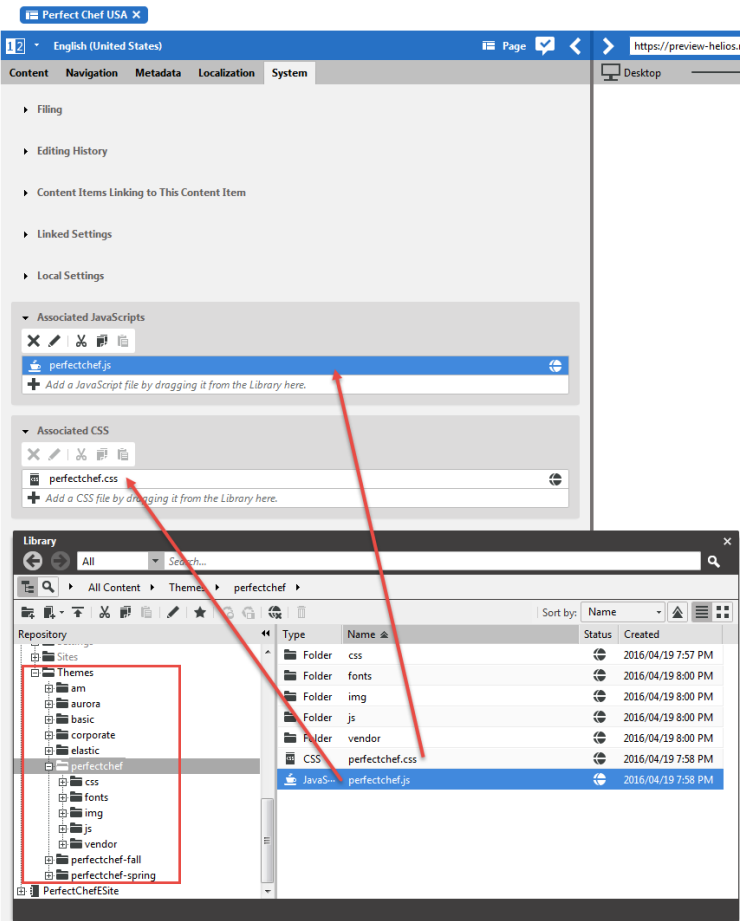


Figure 6.25. Linking a theme to site root

Release work

The release work consists of two parts:

- publishing the content
- exporting the content to your VCS (optional)

Publishing

Just like every other content, web resources imported to the *Content Server* need to be published in order to let the changes affect the live CAE. See the *CoreMedia Studio User Manual* for details about publishing content.

Exporting

If you want to export the resources to a local repository, you need the server export tool. For details of the server export tool consult Section “Serverimport/Serverexport” in *CoreMedia Content Server Manual*.

This can be useful if you want to store content in your local repository, for example to use versioning of the content itself for later re-import into a different *Content Management Server*. You then need to export content created for the web resource and every content you linked the content to.

Keep in mind that if you have dependencies inside your web resources, for example you have a CSS file which styles the background of an element using a background image, you need to export the created content for the image file as well. Keep in mind that the content is just exported to the local file system so you need to commit the created files to the repository afterwards.

6.5 Localized Content Management

One of the primary challenges when engaging in a global market is to reach all customers in different countries.

The first most obvious task is to provide your website contents in different languages. But in addition you may also want to customize your advertised products to local holidays or meet the different legal requirements in different countries.

CoreMedia DXP 8's Multi-Site concept assists you in meeting these requirements.

6.5.1 Concept

There are many possible approaches to fulfill the requirements for providing multiple sites in different countries. *CoreMedia DXP 8* offers a solution which you can customize to your needs and to the workflows you are used to.

The following chapter will present the basic ideas and concepts of *CoreMedia DXP 8's* Multi-Site to you.

Terms

The multi-site concept and documentation is based on the following terms. You may skip this section for now and return to it later when these terms are referenced.

Locale

The term locale refers to the concept of translation and localization. Thus, it is in general a combination of a country and a language. So if the country *Switzerland* requires contents to be available in English, Italian, German and Romansh, four locales have to be defined.

locale = country + language

The locale is represented as IETF BCP 47 language tag (*Tags for Identifying Languages*).

IETF BCP 47

Site

A site is a cohesive collection of web pages in a single locale, sometimes referred to as localized site. Technically a site consists of:

- ➔ the site folder,
- ➔ the site indicator,
- ➔ the site's home page and

- other contents of the site.

Master Site

A master site is a site other localized sites are derived from. A localized site might itself take the role of a master site for other derived sites. This reflects the need that your localized Canadian site (which is in English) needs another localized variant in French.

Site Folder

All contents of a site are bundled in one dedicated folder. A typical example of a site folder is:

```
/Sites/MySite/Canada/French
```

Site Indicator

A site indicator is the central configuration object for a site. It is an instance of the content type `CMSite`. It explicitly configures:

- the site's home page,
- the site identifier,
- the site name,
- the site's locale,
- the master site and
- the site manager group.

It also implicitly defines the root of the site folder.

Home Page

The site's home page is the main entry point for all visitors of a site. Technically it is also the main entry point to calculate the default layout and the contents of a site.

Site Identifier

The site identifier needs to be unique among all sites. It can be used to reference a site reliably also outside the CMS for example in configuration files.

Site Name

The site name is the name of a master site and all derived sites. A derived site inherits

Site Manager Group

the site name from its master site and must not change it.

Members of a site manager group are typically responsible for one localized site. The recommendation is to have one dedicated group for each site with appropriate permissions applied for the site folder.

Responsible means that they take care of the contents of that site and that they accept translation tasks for that site. If a site manager is allowed to also trigger translation tasks from the master site to their site they need to be added to the translation manager role.

While the Site Manager Groups are typically local to their site, thus represent the horizontal layer, *CoreMedia Digital Experience Platform 8* also introduces a vertical layer referred to as *global site manager*. As a consequence the members of the horizontal layer are sometimes referred to as *local site managers*.

Global and Local Site Manager

Global site managers have an overview over all sites while local site managers focus on their sites with additionally required access to the particular master site for translation processes.

Translation Manager Role

Editors in the translation manager role are in charge of triggering translation workflows either from or to a site.

Sites Structure

CoreMedia DXP 8 assumes that your localized sites are all derived from one master site. The site hierarchy might be nested, thus a site derived from the master site again might have derivatives. You can trigger the localization process from your master site, directly derived sites will adapt and forward changes to their derived sites.

The examples below refer to the default configuration which comes with *CoreMedia Blueprint*. To adapt the structure to your needs you have to configure the `SiteModel` - see also [Section “Site Model and Sites Service” \[342\]](#).

Multi-Site Folder Structure

All elements belonging to a site structure are placed in one dedicated folder. In this folder you will find the master site as well as all derived localized sites.

Another set of master and derived sites could be created in parallel to that site following the same concept.

```
/Sites/  
  MySite/  
    United States/  
      English/ master  
      Spanish/ derived from U.S. English  
    Canada/  
      English/ derived from U.S. English  
      French/ derived from Canadian English  
  MyOtherSite/ another master site structure
```

Example 6.18. Multi-Site Folder Structure Example

The folder structure of the master site and its target sites should be kept equal to avoid the automatic recreation of removed or renamed folders during the translation workflow.

In addition to this common aspects for all sites might be placed outside this folder structure. For details see [Section 6.3.1, "Folder and User Rights Concept" \[264\]](#).

Site Folder Structure

The central entry point into the site folder is the site indicator. It points implicitly to the site's root folder (as it needs to be located at the same folder hierarchy depth among all sites in the system) and points explicitly to the site's home page.

Assuming that your site indicator is always placed in some folder like `Navigation` your site folder structure may look like this:

```
MySite/  
  United States/  
    English/  
      Navigation/  
        MySite [Site] site indicator  
        MySite site's home page  
        ...
```

Example 6.19. Site Folder Structure Example

While the above describes the mandatory folder structure for a site, there are additional structures which adhere to the proposed separation of concerns in [Section 6.3.1, "Folder and User Rights Concept" \[264\]](#), thus within a site you can have several user roles taking care of different aspects of the site as there are:

- ➔ **Editorial content:** For example, articles, images, collections etc. This is the real content of a site that is rendered to the web page. They are located in folders `Editorial`, `Pictures` and `Videos`.
- ➔ **Navigation content:** Channels that span the navigation tree and provide context information, as well as their page grids (see also [Section 6.3.2, “Navigation and Contexts” \[265\]](#)). These contents are located in a folder named `Navigation`.
- ➔ **Technical content:** Site specific, technical documents, like actions, settings, view types, etc. They can be found in folder named `Options`.

Site Interdependence

Having a site derived from its master you will have two layers of interdependence:

1. The site indicator points to its master site indicator.
2. Each derived document points to its master annotated by the version of the master when the derived document retrieved its last update from the master. This information is used in the update process when a new master version requires its derived contents to be translated again.
3. A site indicator inherits the site name from its master. If a site indicator has no master it has to define the site name, which will be used for all derived sites.

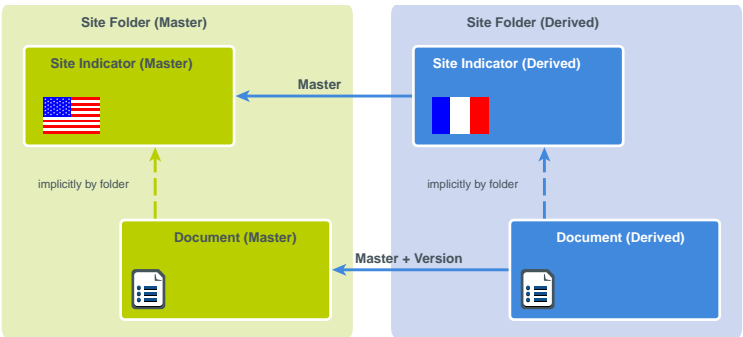


Figure 6.26. Multi-Site Interdependence

The `master` property is configured as weak link by default. Thus, you might publish derived sites before (or without) publishing the master site.

Modifying the Site Structure

Whenever possible, the structure of a site should not be changed after it has been set up initially. In particular, you should not:

- Change the id of a site. If you do so, you must at least re-index its entire master site, if any. See Section 5.2, “Configuring the CAE Feeder” in *CoreMedia Search Manual* for details on the re-indexing procedure. However, the site id might also be stored in other places that a simple re-indexing will not update.
- Move a content to a different site. If you do so, you must at least update the master links of the affected contents to point into the master site of the new site.
- Change the locale of a site. If you do so, you must at least update the locale stored in each individual content of the site.
- Change the master site of a site. If you do so, you must at least update the master links of all contents in the site.

After significant changes of the site structure, you should run the `cm validate-multisite` tool to detect inconsistencies in the content. See Section “Validate Multi-Site” in *CoreMedia Content Server Manual* for details.



6.5.2 Administration

Using *CoreMedia DXP 8's* Multi-Site concept requires some administrative efforts which are described in this section.

Locales Administration

Each site is bound to a specific locale (see [Locale \[332\]](#)). In order to ensure a consistent usage of locale strings across multiple sites that might be managed in a single content repository, the entire list of available locales is maintained in a central document of type `CMSSettings`.

The document `/Settings/Options/Settings/LocaleSettings` contains in the property `Settings` a String List property `availableLocales` which contains locale strings. [Example 6.20, “XML of locale Struct” \[337\]](#) shows the XML structure of the Struct:

LocaleSettings document for locale configuration

Example 6.20. XML of locale Struct

```
<settings>
  <Struct xmlns="http://www.coremedia.com/2008/struct"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <StringListProperty Name="availableLocales">
      <String>de</String>
    </StringListProperty>
  </Struct>
</settings>
```

Please make sure, that the path to the `LocaleSettings` is configured in the Studio properties, as described in Section 7.15, “Available Locales” in *CoreMedia Studio Manual*.

For providing a new locale, you can simply open the document `LocaleSettings` and add a new entry to the list of locales. See Section 4.6.4, “Editing Struct Properties” in *CoreMedia Studio User Manual* for details on how to edit a struct property and add items to string lists. Figure 6.27, “Locales Administration in CoreMedia Studio” [338] shows a Studio tab in which the `LocaleSettings` document is being edited.

Sometimes you might want to define locales for a supranational region such as Africa or Latin America. In this case you can add the language code followed by the UN M.49 area code as described in http://en.wikipedia.org/wiki/UN_M.49. For Spanish in Latin America and the Caribbean add, for example, “es-419”.

Supranational regions

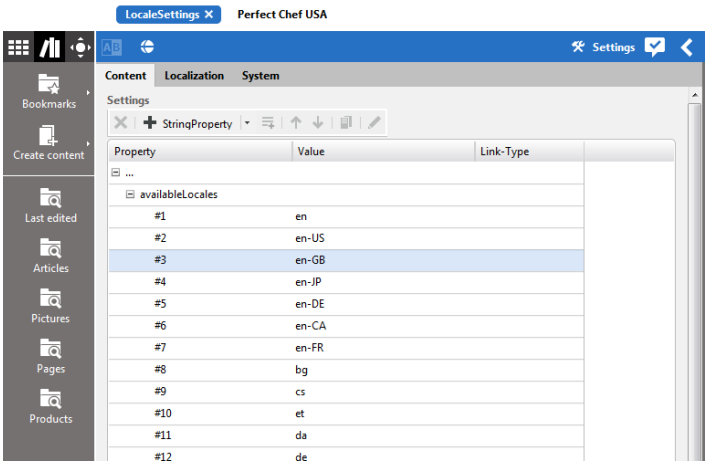


Figure 6.27. Locales Administration in CoreMedia Studio

Groups and Rights Administration

This chapter describes all groups and users, that have to be defined for localization. There are several explicit groups and one user, that can be configured in the `SiteModel` - see also Section “Site Model and Sites Service” [342]. For an overview of predefined editorial users that come with *CoreMedia Blueprint* have a look at Appendix - Predefined Users [484].

The translation manager role is defined once in the property `translationManagerRole` of the `SiteModel`. It is a required group for every user that needs to start a translation workflow and to derive a site.

translation manager role

In case, you do not want to allow every translation manager to also derive sites, it is advisable to create an additional global site manager group, that has the right, to make modifications in the global sites folder.

global site manager

Members of a site manager group take care of the contents of one or more sites. They may for example accept translation workflows if they manage the corresponding target site of a workflow. Or they may start a translation workflows from the master site. For the latter, they must also be member of the translation manager role group, which is described above.

site manager group

The site manager group can be defined in the site indicator. The name of the corresponding property field is defined in the `siteManagerGroupProperty` of the `SiteModel`. If not specified, the group "administratoren" will be used by default. This is also the fallback if the defined group is not available.

There are two ways to set the site manager group:

- While deriving a new site in the sites window, you can set the group.
- Directly in the site manager group property of the site indicator.

For technical reasons the actual changes during a translation workflow are performed as the translation workflow robot user as configured in the property `translationWorkflowRobotUser` of the `SiteModel`. The user needs read and write access on the sites taking part in a translation workflow. As this user is only technical, access to the editor and filesystem services should be restricted, which can be done in the file `jaas.conf` in the module `content-management-server-webapp`. (For details see Section "LoginModule Configuration in `jaas.conf`" in *CoreMedia Content Server Manual*).

translation workflow robot user

Overview of required users and groups for multi-site

Table 6.18, "Suggested Users and Groups for multi-site" [339] shows an example, how the configuration of user groups may look like in *CoreMedia Blueprint*.

Type	Name	Member of	Rights	Remark
group	global-site-manager	approver-role, publisher-role, translation-manager-role	<div>→ /Home (folder: RMDS, content: RSF)</div> <div>→ /Settings (folder and content: R)</div> <div>→ /System (folder and content: R)</div> <div>→ /Sites (folder: RAPSf, content: RMDAPS)</div>	

Table 6.18. Suggested Users and Groups for multi-site

Type	Name	Member of	Rights	Remark
group	local-site-manager	approver-role, publisher-role, translation-manager-role	→ /Home (folder: RMDS, content: RSF) → /Settings (folder and content: R) → /System (folder: RF, content: RMD) → /Themes (folder: RAPF, content: RMDAP) → /Sites/<master-site-root-folder> (folder and content: R)	
group	manager-<language-tag>	local-site-manager	→ /Sites/<site-root-folder> (folder: RAPF, content: RMDAP)	Suggested pattern configured in siteManagerGroupPattern of the SiteModel
group	translation-manager-role			Configured in translationManagerRole of the SiteModel
group	translation-workflow-robots		→ / (folder and content: R) → /Sites (folder: RF, content: RMD)	
user	translation-workflow-robot	translation-workflow-robots		Configured in translationWorkflowRobotUser of the SiteModel

The rights abbreviations denote:

→ R - read

→ M - modify / edit

- D - delete
- A - approve
- P - publish
- F - folder
- S - supervise

For further information about the rights, please refer to chapter "User Rights Management" in [CoreMedia Content Server Manual].

Definition while deriving site

When deriving a new site, a proposal for the name of the site manager group is generated from a predefined pattern. By default, the name starts with *manager* followed by the language tag of the selected target locale (see also [Figure 6.28, "Derive Site: Setting site manager group" \[341\]](#)). This pattern may be configured in the property `siteManagerGroupPattern` of the `SiteModel`.

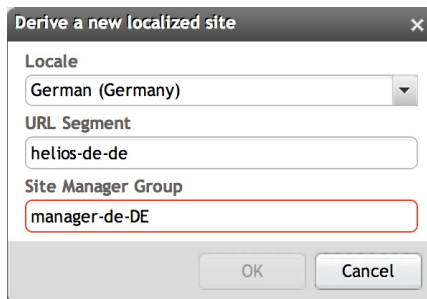


Figure 6.28. Derive Site: Setting site manager group

Adapting site manager group later on

If the site already exists, the name of the site manager group can be set or modified directly in the site indicator (see [Figure 6.29, "Site Indicator: Setting site manager group" \[342\]](#)).

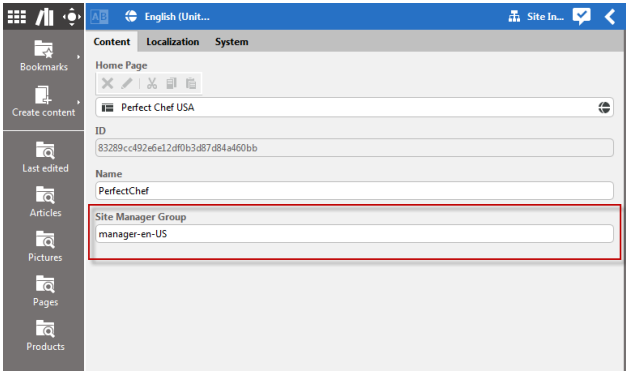


Figure 6.29. Site Indicator: Setting site manager group

If the group does not exist, the property field will be marked red and the creation of the site or the assignment of the group may not be performed, thus the group needs to have been created before. Read more about users, groups and administration in Section 3.16, “User Administration” in *CoreMedia Content Server Manual*.

6.5.3 Development

CoreMedia DXP 8’s Multi-Site concept contains an example implementation for translation and localization processes. As you might have different requirements, for example defined by a translation agency which does the translation for you, the Multi-Site feature is highly configurable. Read the following sections to learn about the configuration options.

Site Model and Sites Service

The site model and the sites service are strongly connected with each other. While the site model consists of properties defining the site structure, the sites service uses this model to work with sites programmatically.

Sites Service

The sites service is designed to access the available sites and to determine the relation between sites and contents. The site model configures the behavior of the sites service.

For developing multi-site features the main entry point is the sites service.

Site Model

The site model is the centralized configuration of the CoreMedia Multi-Site behavior. Its configuration is required in several applications which are listed below. While [section Site Model Properties \[343\]](#) lists the configurable properties (exception:

CoreMedia Site Manager with an extra description in [Site Model in CoreMedia Site Manager \[347\]](#)) the different configuration locations are explained per application:

- [Site Model in CoreMedia Studio \[346\]](#)
- [Site Model in Content Application Engine \[346\]](#)
- [Site Model in Command Line Tools \[347\]](#)
- [Site Model in CoreMedia Site Manager \[347\]](#)

Site Model Properties

The following table illustrates the configurable site model properties. To get to know more about the properties and patterns used, consult the Javadoc of `com.coremedia.cap.multisite.SiteModel`.

Table 6.19. Properties of the Site Model

sitemodel.site.indicator.documentType	
Description	Specifies the content type of the site indicator document. Each site must only have one instance of that content type.
Default Value	CMSite
sitemodel.site.indicator.depth	
Description	Defines the depth under the root of the site folder, where the site indicator document resides.
Default Value	1
sitemodel.site.indicator.namePattern	
Description	Name pattern, which will be used for the name of the site indicator document when deriving a site. Only placeholder {0} is available for this property. For an overview of placeholders see Table 6.20, "Placeholders for Site Model Configuration" [345] .
Default Value	{0} [Site]
sitemodel.site.rootdocument.namePattern	
Description	Defines the pattern for the site's home page document name, used while deriving a site. Only placeholder {0} is available for this property. For an overview of placeholders see Table 6.20, "Placeholders for Site Model Configuration" [345] .
Default Value	{0}
sitemodel.site.manager.groupPattern	
Description	Defines the pattern for responsible default site manager group name when deriving a site. For available placeholders see Table 6.20, "Placeholders for Site Model Configuration" [345]

Default Value	<i>manager-{4}</i>
sitemodel.siteManagerGroupProperty	
Description	Defines the property of the site indicator document holding the site manager group name.
Default Value	<i>siteManagerGroup</i>
sitemodel.translationManagerRole	
Description	Defines the group name denoting the role which permits a user to start a translation workflow.
Default Value	<i>translation-manager-role</i>
sitemodel.idProperty	
Description	Defines the property of the site indicator document which contains the site id.
Default Value	<i>id</i>
sitemodel.nameProperty	
Description	Defines the property of the site indicator document which contains the site name.
Default Value	<i>name</i>
sitemodel.localeProperty	
Description	Defines the property of translatable content and the site indicator document, which holds the locale of the content.
Default Value	<i>locale</i>
sitemodel.masterProperty	
Description	Defines the property of translatable content and the site indicator, which contains the link the master document.
Default Value	<i>master</i>
sitemodel.masterVersionProperty	
Description	Defines the property of translatable content, which contains the version the corresponding master document.
Default Value	<i>masterVersion</i>
sitemodel.rootProperty	
Description	Defines the property of the site indicator document, which refers to the home page document of this site.
Default Value	<i>root</i>
sitemodel.uriSegmentProperty	

Description	Defines the property of the site's home page content type, which defines the root URI segment of the site.
Default Value	<i>segment</i>
sitemodel.uriSegmentPattern	
Description	Defines the pattern for the default root URI segment when deriving a site. For available placeholders see Table 6.20, "Placeholders for Site Model Configuration" [345] .
Default Value	<i>{0}-{4}</i>
sitemodel.rootFolderPathPattern	
Description	Defines the pattern to determine the site folder for a new derived site. For available placeholders see Table 6.20, "Placeholders for Site Model Configuration" [345] .
Default Value	<i>/Sites/{0}/{6}/{5}</i>
sitemodel.rootFolderPathDefaultCountry	
Description	Defines the folder name for the country folder, if the locale chosen while deriving a site defines no country explicitly.
Default Value	<i>NO_COUNTRY</i>
sitemodel.translationWorkflowRobotUser	
Description	Defines the user name of the user responsible for creating derived content during a translation workflow. ➡ The user should have read / write access on all localizable Sites. ➡ The user should not be allowed to use the editor and filesystem services.
Default Value	<i>translation-workflow-robot</i>

Site Model Placeholders

Placeholder	Description	Example
{0}	site name	<i>MySite</i>
{1}	site locale's language code	<i>en</i>
{2}	site locale's country code (defaults to language code, if not available)	<i>US</i>

Table 6.20. Placeholders for Site Model Configuration

Place-holder	Description	Example
{ 3 }	site locale's variant (defaults to country or language code, if not available); using BCP 47 Extensions	<i>u-cu-usd</i>
{ 4 }	site locale's IETF BCP 47 language tag	<i>en-US-u-cu-usd</i>
{ 5 }	site locale's language display name (localized in U.S. English); only available for <code>sitemodel.rootFolderPathPattern</code>	<i>English</i>
{ 6 }	site locale's country display name (localized in U.S. English); only available for <code>sitemodel.rootFolderPathPattern</code>	<i>United States</i>
{ 7 }	site locale's variant with the prefix <code>variantPrefix</code> configured in site model's Spring context; defaults to empty String; only available for <code>sitemodel.rootFolderPathPattern</code> . See IANA Language Subtag Registry for valid registered variants.	<i>_arevela</i>

Application Configurations

For details of the configuration in every application, please read the documentation below.

CoreMedia Studio

The site model default properties can be adjusted in the `application.properties` file in the `src/main/webapp/WEB-INF` directory of the `studio-webapp` module. See Chapter 3, *Deployment* in *CoreMedia Studio Manual* for further information.

Content Management Server

The site model default properties can be adjusted in the `application.properties` file in the `src/main/webapp/WEB-INF` directory of the `content-management-server-webapp` module. See Section 3.1, “Structure of Content Server Installation” in *CoreMedia Content Server Manual* for further information.

Content Application Engine

The site model default properties can be adjusted in the `component-blueprint-cae.properties` file in the `src/main/resources/META-INF/coremedia` directory of the `cae-base-component` module. Thus, the configuration applies to the Live CAE as well as to the Preview CAE. See *CoreMedia Content Application Developer Manual* for further information.

Command Line Tools

The site model default properties can be adjusted in the `commandline-tools-sitemodel.properties` file in the `properties/corem` directory of the `cms-tools-application` module.

CoreMedia Site Manager

The *Site Manager* provides only rudimentary support of the multi-site features especially for backwards compatibility to old CoreMedia systems. For the full set of features please use *CoreMedia Studio*.

To migrate from existing multi-site features of *Site Manager* you need to adapt the `editor.xml` for example by adding a `SiteModel`.

[Example 6.21, “SiteModel in editor.xml” \[347\]](#) shows an example for adding the `SiteModel` to `editor.xml`.

```
<Editor>
  <!-- ... -->
  <SiteModel
    siteIndicatorDocumentType="CMSite"
    siteIndicatorDepth="1"
    idProperty="id"
    rootProperty="root"
    masterProperty="master"
    localeProperty="locale"/>
  <!-- ... -->
</Editor>
```

Example 6.21. SiteModel in editor.xml

Mind that for changing property names of `master` and `masterVersion` you also need to adapt property editors for the versioned master reference as shown in [Example 6.22, “Versioned Master Link in editor.xml” \[347\]](#).

```
<Document type="..." viewClass="...">
  <!-- ... -->
  <Property
    name="master"
    editorClass="hox.corem.editor.toolkit.property.VersionLinkEditor"

    versionProperty="masterVersion"/>
  <Property
    name="masterVersion"
    editorClass="hox.corem.editor.toolkit.property.InvisibleEditor"/>

  <!-- ... -->
</Document>
```

Example 6.22. Versioned Master Link in editor.xml

Content Type Model

While you might create your very own content type model, the following description is based on the assumption that you use the content type model of *CoreMedia*

Blueprint. For a custom content type model you must meet certain requirements which are described at the end of this section.

Content Types

The base content type for any contents which require to be translated is `CMLocalized`. For further information see [Section 9.6, “Content Type Model” \[476\]](#).

```
<DocType Name="CMLocalized" Parent="CMObject" Abstract="true">
  <StringProperty Name="locale" Length="64"/>
  <LinkListProperty Name="master" Max="1"
    LinkType="CMLocalized"
    extensions:weakLink="true"/>
  <IntProperty Name="masterVersion"/>
  ...
</DocType>
```

Example 6.23. *CMLocalized*

Weak Link Attribute

The contents of each site have to be published and withdrawn independently of their master. Therefore, the `weakLink` attribute of every master property must be set to true - see also Content Type Model - `LinkListProperty` in *CoreMedia Content Server Manual*.

Translatable Properties

The properties that have to be translated when in derived sites are marked as translatable in the content type model by attaching the `extensions:translatable` attribute to the property declaration - see also Content Type Model - Translatable Properties in *CoreMedia Content Server Manual*.

```
<DocType Name="CMTeasable" Parent="CMHasContexts" Abstract="true">
  <LinkListProperty Name="master" Max="1"
    LinkType="CMTeasable"
    Override="true"
    extensions:weakLink="true"/>
  <StringProperty Name="teaserTitle" Length="512"
    extensions:translatable="true"/>
  <XmlProperty Name="teaserText" Grammar="coremedia-richtext-1.0"
    extensions:translatable="true"/>
  <XmlProperty Name="detailText" Grammar="coremedia-richtext-1.0"
    extensions:translatable="true"/>
  ...
</DocType>
```

Example 6.24. *CMTeasable*

Automatically Merged Properties

Usually all non-translatable properties in the master content will be applied automatically to the derived content when a translation task is accepted. This helps to keep binary and structural data in sync between sites, such as images, crops, settings, and the navigation hierarchy, and complements the XLIFF-based update of translatable properties. To enable the automatic merge for a translatable property

or disable the automatic merge for a non-translatable property the `extensions:automerge` attribute has to be attached.

```
<DocType Name="CMSSettings" Parent="CMLocalized">
  <LinkListProperty Name="master" Max="1" LinkType="CMSSettings"
  Override="true" extensions:weakLink="true"/>
  <XmlProperty Name="settings" Grammar="coremedia-struct-2008"
  extensions:translatable="true" extensions:automerge="true"/>
  <StringProperty Name="identifier" Length="100"/>
</DocType>
```

Example 6.25. *CMSSettings*

Custom Content Type Models

Even if it is not recommended, you can use your own content type model with the Multi-Site feature of *CoreMedia DXP 8*. Prerequisite is, that you can configure the Site Model mentioned before to meet the requirements of your own content type model. In addition, you probably need to adapt your document type model to fit the requirements of the multi-site concept.

Therefore, every content type, which may occur in a site must contain all properties, listed below.

- ➔ master
- ➔ masterVersion
- ➔ locale

Please adapt the configuration of each property to the properties of `CMLocalized` in the example above.

ServerImport and ServerExport

Both `serverimport` and `serverexport` have a special handling built in for the `master` and `masterVersion` properties. The export will store the translation state of a derived document and on import efforts are taken to reestablish a comparable translation state.

For the concrete names of the `master` and `masterVersion` properties, the `SiteModel` has to be provided to the tools, which is done via *Spring* in the file `COREM_HOME/properties/corem/serverimportexport-context.xml` (In *CoreMedia Blueprint* this file is added in the `cms-tools-application` module).

Examples:

The examples assume that you export a document and its master and import it afterwards into a clean system. The table uses # (hash mark) to denote contents having the given latest version number.

Master (before)	The master version before export. <i>none</i> means that no master link is set.
Version (before)	Value of the master version property of the derived document before export. <i>none</i> means that no version is specified yet which actually marks derived documents as not being up to date with its master document.
State (before)	The translation state of the derived document before export.
Master (after)	The master version after import (actually always #1).
Version (after)	Value of the master version property of the derived document after import.
State (after)	The translation state of the derived document after import.

Mas-ter (be-fore)	Ver-sion (be-fore)	State (be-fore)	Mas-ter (after)	Ver-sion (after)	State (after)	Comment
#5	5	up-to-date	#1	1	up to date	The master and derived document were up to date before export (derived document is most recent localization of its master). Thus, after import the same state is set.
#5	4	not up-to-date	#1	0	master ver-sion des-troyed	The master and derived document were not up to date before export. Thus, after import the value of the master version property is set to a special version number denoting that the derived content is not up-to-date. On API level this is regarded as if the referred master version got destroyed meanwhile. For the editor the document will appear as being not up-to-date.

Table 6.21. Example for server export and import for multi-site

Master (before)	Version (before)	State (before)	Master (after)	Version (after)	State (after)	Comment
#5	none	not translated yet	#1	none	not translated yet	Derived document was never localized from its master. Thus, the same state applies after import.
none	5	no master	none	5	no master	Corrupted content: No special logic is applied. The overall approach for import and export is defensive thus if the state was invalid before, the fallback is to use the default behavior from import and export keeping the values as is.

XLIFF Integration

Translation jobs can be represented using the XLIFF, the *XML Localization Interchange File Format*. XLIFF is an OASIS standard to interchange localizable data tools as for example used by translation agencies. An XLIFF file contains the source language content of translatable properties from one or more documents. It is then enriched by a translation agency to contain the translated content, too. *CoreMedia DXP 8* support XLIFF 1.2, where version 1.2 is the most recent final specification.

An XLIFF file is structured into multiple translation units. While a string property is encoded as a single translation unit, a richtext property is split into semantically meaningful parts, comprising for example a paragraph or a list item. Translation units are then grouped, so that units belonging to a single property are readily apparent.

All properties of a single document are included in a single file section according to the XLIFF standard. A custom attribute allows the importer to identify the target document that should receive the translation, as supported by the XLIFF standard. Translation tools must preserve this extension attribute when filling the target content into the XLIFF file.

The following fragment shows the start tag of a <file> element for translating from English to French, indicating the source document 222 and the target document 444.

```
<file
  xmlns:cmxliiff=
    "http://www.coremedia.com/2013/xliiff-extensions-1.0"
```

Example 6.26. XLIFF fragment

```
original="coremedia:///cap/version/222/1"
source-language="en"
datatype="xml"
target-language="fr"
cmxliiff:target="coremedia:///cap/content/444">
```

Translation Workflow

Translation Workflow Configuration

This section describes general configuration options for translation workflows.

XLIFF

The handling of empty translation units during XLIFF import can be configured using the following properties:

Table 6.22. XLIFF Properties

Property Name	Description	Default Value
translate.xliff.import.emptyTransUnit-Mode	Configure handling of empty trans-unit targets for XLIFF import. Possible values: ➔ <i>IGNORE</i> : Empty targets are allowed. On import the empty translation unit will replace a possibly non-empty target and thus delete its contents. <i>FORBIDDEN</i> : No empty targets are allowed <i>IGNORE- NORE_WHITE- SPACE</i> : Empty targets are only allowed where the matching source is empty or contains only whitespace characters	<i>IGNORE_WHITE- SPACE</i>
translate.xliff.import.ignorable-WhitespaceRegex	Configure the regular expression that determines which characters are counted as ignorable whitespace. This configuration is only used when trans-	<code>[\\s\\p{Z}]*</code>

Property Name	Description	Default Value
	late.xliff.import.emptyTransUnit-Mode is set toIG-NORE_WHITE-SPACE.	

Translation Workflow Studio UI

The translation workflow UI in *CoreMedia Studio* consists of several panels that enable site managers to start, view and control a translation workflow. Like the workflows themselves (see [section “Configuration and Customization” \[364\]](#)), these panels are also highly customizable.

To get a glance of the Workflow UI see Section 2.4.2, “Control Room” in *CoreMedia Studio User Manual* and Section 4.7.3, “Translating Content” in *CoreMedia Studio User Manual*.

addTranslationWorkflowPlugin

The configuration of the workflow panels is done in the `ControlRoomStudioPlugin.xml` using the `com.coremedia.cms.editor.controlroom.config.addTranslationWorkflowPlugin`. The `addTranslationWorkflowPlugin` configures four separate panels for one `processDefinitionName`:

→ `startPanel:com.coremedia.cms.editor.controlroom.config.abstractStartTranslationWorkflowPanel`

The panel to start one or more translation workflows. It creates the `Process` instances, sets process variables and finally starts the process.

→ `inboxPanel:com.coremedia.cms.editor.controlroom.config.workflowForm`

The panel to display and control a `Task` in the user's *inbox*.

→ `pendingPanel:com.coremedia.cms.editor.controlroom.config.workflowForm`

The panel to display a `Process` in the user's *pending* list.

→ `finishedPanel:com.coremedia.cms.editor.controlroom.config.workflowForm`

The panel to display a finished `Process` in the user's *finished* list.

workflowForm

As you can see, the type of all these panels except the `startPanel` is `com.coremedia.cms.editor.controlroom.config.workflowForm`. This

is the base component for displaying `Processes` and `Tasks` and it is inspired by the `com.coremedia.cms.editor.sdk.config.documentForm`. Like the `documentForm` the `workflowForm` has some configuration options that are provided to the form by the framework and are forwarded to nested items using component defaults:

→ `bindTo`

A value expression returning the process to show.

→ `bindToTask`

A value expression returning the task to show. If this form displays a process and not a task (especially in the finished list) this option is null.

→ `forceReadOnlyValueExpression`

An optional `ValueExpression` which makes the component read-only if it is evaluated to true.

→ `processDefinitionName`

The name of the definition of processes that may be displayed using this form. This configuration option is not forwarded to nested items.

For detailed information consult the [CoreMedia Studio ActionScript API](#).

Customization

Most panels that are used by default only have very few configuration options. This means that if you want to customize these, you will likely have to implement your own `workflowForms` or start panel.

The only exception is the `defaultTranslationWorkflowDetailFrom`, which can be configured with the state transitions of the workflow. When you have a look at the provided translation workflow (see [Section 6.6.2, “Predefined Translation Workflow” \[362\]](#)), you will notice a process variable called `translationAction`. This variable is used to let the user select the next workflow step, which works by setting the value of the `translationAction` variable to the selected value of the radio group shown in the form. A click on the **[Apply]** button will then complete the `Translate` task, which is followed by a `Switch` task that maps the value of the `translationAction` to a successor task.

So, by adding a new `Switch` case and a new successor task for example, you can easily create another translation option. The mapping from UI to workflow is done with the `workflowStateTransitions` configuration option of the `defaultTranslationWorkflowDetailFrom`. This is basically a map from the current value of the `translationAction` called `state` to a list of possible next values for the `translationAction` called `successors`. Each `workflowStateTransition` creates a radio group with the `successors` as radio buttons, that will be displayed, when the current value of the `translationAction` matches the `state`.

Localization

Localization of the translation workflow UI is done in the file `BlueprintProcessDefinitions.properties`, which uses the following patterns:

- `<ProcessDefinition-name>_text`: the name of the `ProcessDefinition`
- `<ProcessDefinition-name>_state_<state>_text`: the name of a state or successor of a `workflowStateTransition`
- `<ProcessDefinition-name>_task_<task>_text`: the name of a `Task`

6.6 Workflow Management

In this chapter you will find a description of the predefined workflows as well as the workflow actions that are needed to customize existing workflows or define new ones.

Predefined workflows described in here:

- workflows covering the publication of resources, see [Section 6.6.1, “Publication” \[356\]](#),
- an example translation workflow, see [Section 6.6.2, “Predefined Translation Workflow” \[362\]](#),
- a fixed workflow for initially deriving a site from an existing site, see [Section 6.6.3, “Deriving Sites” \[369\]](#).

6.6.1 Publication

In this chapter you will find a description of publication workflows and a description of the publication semantics.

CoreMedia delivers the listed example workflows. But the workflow facilities are not restricted to those features. They can be tailored to fit all types of business processes.

Approval and Publication of Folders and Content Items

A publication synchronizes the state of the *Live Server* with the state of the *Content Management Server*. All actions such as setting up new versions, deleting, moving or renaming files, withdrawing content from the live site require a publication to make the changes appear on the *Live Server*.

What is and what does a publication?

CoreMedia makes a distinction between the publication of structural and of content changes:

- Content-related changes are changes in document versions such as a newly inserted image, modified links, text.
- Structure-related changes are moving, renaming, withdrawing or deleting of resources. So it becomes possible to publish structural changes separately from latest and approved document versions.

For every publication a number of changes is aggregated in a change set. This change set is normally composed in the course of a publication workflow. The administrator and other users with appropriately configured editors can also execute a direct publication, which provides a simpler, although less flexible means of creating a change set.

Change Set in Direct Publications

When performing a direct publication, the change set is primarily based on the set of currently selected resources or on the single currently viewed resource. As the set of resources does not give enough information for all possible types of changes, three rules apply:

- You cannot publish movements and content changes separately. Whenever applicable, both kinds of changes are included in the change set.
- When a document is marked for deletion or for withdrawal, new versions of that document are not published.
- If the specific version to be published is not explicitly selected, the last approved resource version is included in the change set.

There are also some automated extension rules for the change set, which modify the set of to-be-published resources itself. These rules can be configured in detail. Ask your Administrator about the current settings.

- When new or modified content is published and links to an as yet unpublished resource, the unpublished resource is included in the change set. Depending on the configuration, also recursively linked documents can be included in the change set. Target documents that are linked via a weak link property are not included in the change set.
- When the deletion of a folder is published, all directly and indirectly contained resources are included in the change set.
- When the withdrawal of a folder is published, all directly and indirectly contained published resources are included in the change set.
- When the creation, movement, or renaming of a resource in an unpublished parent folder is published, that folder is included in the change set.

Preconditions

Preconditions for a successful publication

Preconditions for a successful publication are:

- all path information concerning the resource has to be approved too: if the resource is located in a folder never published before, this folder has to be published with the resource. So, add it to the change set or publish the folder before.
- withdrawals and deletions must be approved before publication.
- all documents linked to from a document which is going to be published have to be already published or included in the change set. This is because a publication that would cause dead links will not be performed. This rule does not apply for weak link properties.
- a document which is going to be deleted must not be linked to from other documents or these documents have to be deleted during the same publication. This rule does not apply for weak link properties.

Table 6.23. Publishing documents: actions and effects

Status and action on the <i>Content Management Server</i>	Effect on the <i>Live Server</i> on publication
A version of the document does not yet exist on the <i>Live Server</i> . The document is not marked for deletion. You approve the version.	The approved version is copied to the <i>Live Server</i> .
The last approved version of a document already exists on the <i>Live Server</i> . The document is not marked for deletion. You start a new publication without any further preparation.	No effect on the <i>Live Server</i> .
The document is published and is not marked for deletion. It therefore exists on both servers. You rename the document and approve the change.	The document is renamed.
The document is published and is not marked for deletion. It therefore exists on both servers. You move the document and approve the change.	The document is moved.
The document is published. It therefore exists on both servers. No links to this document exist. You mark the document for withdrawal and approve the change.	The document is destroyed on the <i>Live Server</i> .
The document is published. It therefore exists on both servers. No links to this document exist. You mark the document for deletion and approve the change.	The document is destroyed on the <i>Live Server</i> . The document is moved into the recycle bin on the <i>Content Management Server</i> .
The document is published. It therefore exists on both servers. Links to this document from other published documents exist. You mark the document for deletion and approve the change.	The deletion cannot be published, since an invalid link would be created. A message is displayed in the publication window. Remove the link in the other document and publish again.
The document is published. It therefore exists on both servers. Weak links to this document from other published documents exist.	The document is destroyed on the <i>Live Server</i> . The document is moved into the recycle bin on the <i>Content Management Server</i> .

Status and action on the <i>Content Management Server</i>	Effect on the <i>Live Server</i> on publication
You mark the document for deletion and approve the change.	

Status and action on the <i>Content Management Server</i>	Effects on the <i>Live Server</i> on publication
The folder is published and is not marked for deletion. It therefore exists on both servers. You rename the folder and approve it.	The folder is renamed.
The folder is published and is not marked for deletion. It therefore exists on both servers. You move the folder and approve the change.	The folder is moved.
The folder is not published and not marked for deletion. You approve the folder.	The folder is created on the <i>Live Server</i> .
The folder is published. You mark it for withdrawal. When queried, you acknowledge the mark for withdrawal of all contained resources. You approve the change.	The folder is destroyed on the <i>Live Server</i> . The withdrawal can only succeed if all resources on the <i>Live Server</i> or <i>Content Management Server</i> that are contained in the folder, and all published resources that link to this folders content via a non-weak link property, are also contained in the change set.
The folder is published. You mark it for deletion. When queried, you acknowledge the mark for deletion of all contained resources. You approve the change.	The folder is destroyed on the <i>Live Server</i> . The folder is moved to the recycle bin on the <i>Content Management Server</i> . The deletion can only succeed if all resources on the <i>Live Server</i> or <i>Content Management Server</i> that are contained in the folder, and all published resources that link to this folders content via a non-weak link property, are also contained in the change set.

Table 6.24. Publishing folders: actions and effects

Special cases

Special cases

Please keep in mind that:

- ➡ Older versions cannot be published.

Example: if a version No. 4 had already been published it is not possible to publish version No. 3 thereafter. To do so, create a version No. 5 from No. 3.

- During a deletion, a resource that has not been published yet is moved to the recycle bin immediately.

In addition, consult the previous tables for effects of a publication depending on the state of the resource. For all examples it is assumed that you have appropriate rights to perform the action.

Withdrawing Publications and Deleting Resources

Delete and withdraw resources

There is only one fundamental difference between withdrawal of publications and deletion: a withdrawal affects only the *Live Server*, whereas the deletion of a resource - folder or document - causes the resource to be moved into the trash folder on the *Content Management Server*.

Before a withdrawal or deletion can be published as described before, a mark for withdrawal or for deletion must be applied using the appropriate menu entries or tool bar buttons. In the case of folders, the contained resources are affected, too. If you have marked a resource for deletion and withdrawal, then the deletion will be executed.

- When a folder is marked for deletion, all contained published resources are marked for deletion, too. Not published resources are immediately moved into the recycle bin without requiring you to start a publication.
- When a folder is marked for withdrawal, all contained published resources are marked for withdrawal, too.
- When a mark for withdrawal or deletion of a folder is revoked, this also affects all contained resources with the same mark.
- If you use direct publication and approve a folder that is marked for withdrawal deletion, that approval is implicitly extended to the contained resources that are also marked for withdrawal or deletion.
- Disapprovals extend to contained resources in the same way.

Predefined Publication Workflows

The predefined workflows for the approval and publication of resources are described in the following table. These workflows can be uploaded using `cm upload -n <filename>`. You can examine their definition and use them as examples for your own definitions, by downloading an uploaded definition using `cm download <ProcessName>`.

Table 6.25. Predefined publication workflow definitions

Workflow	Definition name
simple publication	Process StudioSimplePublication defined in studio-simple-publication.xml
2-step publication	Process StudioTwoStepPublication defined in studio-two-step-publication.xml

Publication workflow steps

The following table compares the working steps which are covered by the pre-defined workflows.

Step	simple publication	2-step publication
1.	A user creates the workflow with all necessary resources.	A user creates the workflow with all necessary resources.
2.	The resources are published (and implicitly approved) in one step, performed by the same user, who needs 'approve' and 'publish' rights.	A second user (needs 'approval' and 'publish' rights) can explicitly approve resources. In <i>Studio</i> , the second user may also modify the resources before
3.		Publication will be executed when finishing the task after all resources in the change set have been approved.
4.		(If not, the workflow is returned to its 'composer')

Table 6.26. Predefined publication workflow steps

Features of the Publication Workflows

The predefined publication workflows have some features in common, which are described in the following:

Users and Groups

In order to execute tasks within workflows, users have to be assigned to special groups. In the predefined publication workflows, these are the following:

- 1. *composer-role*: to be able to create (and start) a publication workflow and compose a change set
- 2. *approver-role*: to be able to approve the resources in the change set
- 3. *publisher-role*: to be able to publish the resources in the change set

Special groups can be defined and linked to the workflow via the `Grant` element in the workflow definition file. Read more about users, groups and administration in the *Content Server Manual*.

Note that, when all eligible users for a task reject that task, the task is again offered to all eligible users. So if you are the only user for an *approver-role* group and you start a publication workflow, the second step of the workflow will be escalated. That is because you cannot be the composer and the approver of a resource - and there is no other user than you.

Basic Steps in a Publication Workflow

After a user has created one or more documents, these documents should be proofread, approved and published in a workflow:

- 1. The user (not necessarily the user who did the editing) starts a workflow. If he selects resources at starting time, these resources will be added to the change set and the compose task will be accepted automatically. Otherwise, he has to add the resources to the change set later.
- 2. The user completes the 'compose' task.
- 3. The task 'approve' is automatically offered to all appropriate users (members of the *approver-role* group, but not to the composer - even if he is a member of this group). Somebody accepts the task and approves the resources.

The user has the following options:

option A	option B	option C	option D
The user accepts the task, approves the resource(s) and finishes the task. All resources are approved.	The user accepts the task, does not approve all resource(s) and finishes the task	The user rejects the task.	The user accepts the task but delegates it to somebody else.
The task 'Publication' is offered to all members of the group <i>publisher-role</i> .	The change set is sent back to the user who completed the 'compose' task.	The task is offered all other members of the group <i>approver-role</i> .	The task is automatically accepted by this user.

Table 6.27. User options.

6.6.2 Predefined Translation Workflow

A translation workflow can be used to communicate changes in the project of a master site to the derived sites.

CoreMedia Blueprint provides one template translation workflow named *Translation* in the file `translation.xml` in the `wfs-tools-application` module. The

workflow is built around an empty action, the `SendToTranslationServiceAction` in the `workflow-lib` module, which is supposed to implement the sending / receiving of contents to / from a translation agency. Without an implementation of this action, the workflow can still be used for manual in-house translation, possibly in conjunction with XLIFF download/upload.

Roles and Rights

The translation workflow process is based on two roles defined for *CoreMedia DXP 8's* Multi-Site concept:

1. The group `translation-manager-role` contains all users that are allowed to start a translation workflow. The name of this group has to be configured in the property `translationManagerRole` of the `SiteModel` (see [section “Site Model” \[342\]](#)). After changing this property, you have to upload the workflows again, because uploading persists the current property value.
2. The site manager group defines the users who may accept translation workflows for the content of a site. [Groups and Rights Administration for Localized Content Management \[338\]](#) describes how to set this property for every site.

Workflow Lifecycle

As described in [section “Roles and Rights” \[363\]](#), the translation managers start the translation workflow for a set of new or changed contents from the Control Room. Therefore, a new `Process` instance will be created for every site that has been selected as a translation target.

At first, the `Process` instances both run two `AutomatedTasks` that retrieve the manager group and collect / create the derived contents for the target site. For details see [Section “Predefined Translation Workflow Actions” \[364\]](#).

The following `UserTask` called `Translate` is used to let the user choose a next step. This is done by selecting a next step in the radio group of the `workflowForm`. The selected value will then be set as value for the `translationAction` process variable. This variable is then used in a `Switch` task to choose the successor task.

These successor tasks are:

- ➔ `SendToTranslationService`: Send / retrieve content to / from translation agency (has to be implemented in the project)
- ➔ `Rollback`: Cancel the translation and rollback changes that may have been made to the target content. (E.g.: The `GetDerivedContentsAction` may have created content in the target site derived from the provided master content.)

- **Complete:** Update the `masterVersion` of the target content to indicate, that the translation is completed. This can be used, for example when the user translated the content manually.

While the `Rollback` and `Complete` tasks finish the process, the `SendToTranslationService` task has another `UserTask` successor called `Review`. This task simply gives the user an opportunity to check the content imported from the translation agency. For details on the Actions behind these tasks see [Section “Predefined Translation Workflow Actions” \[364\]](#).

Configuration and Customization

The example translation workflow is meant to be configured to your needs. You might define multiple translation workflows, like translation via translation agency or manual translation performed by the site managers. The only restriction is that every translation workflow needs a process variable `subject` of type `String`, which will be set by the framework.

In order to reliably track content that is currently "in translation", you also need to define, configure and regularly invoke an instance of the `com.coremedia.translate.workflow.impl.CleanInTranslation` class. An example definition is included in the blueprint source in the `workflowserver-springcontextmanager.xml` file, which you may have to adapt.

Be aware, that changes in the process definition will probably lead to changes in the UI, too. If you want to change only small bits of the provided translation workflow like adding another user-selectable `translationAction` and `Task`, this can be done pretty easily through configuration of the `defaultTranslationWorkflowDetailForm` inside the `ControlRoomStudioPlugin`.

But if want to use a workflow completely different to the one provided, be prepared to write your own implementations of the `workflowForms` and `start panel` used to display your workflow in *Studio*.

For details on customizing workflows see the [CoreMedia Workflow Manual]. For details on customizing the *Studio* UI for the translation workflows see [Section “Translation Workflow Studio UI” \[353\]](#).

Predefined Translation Workflow Actions

This section describes various actions that can be used to define a translation workflow.

- [Section “GetDerivedContentsAction” \[365\]](#) describes an action that computes, and if necessary creates derived contents from a given set of master contents.
- [Section “GetSiteManagerGroupAction” \[366\]](#) describes an action that determines a site manager group and stores it in a process variable.

- ➔ [Section “ExtractPerformerAction” \[366\]](#) describes an action that identifies the user who executes that current task and stores a user object in a process variable.
- ➔ [Section “CompleteTranslationAction” \[367\]](#) describes an action that finishes a manual translation process.
- ➔ [Section “RollbackTranslationAction” \[368\]](#) describes an action that rolls back a translation process, possibly deleting spurious content.

GetDerivedContentsAction

This action retrieves all derived contents from a given list of master contents. If a document already exists in the target site and its `masterVersion` equals to the current version of the master content, it will be ignored for the workflow. Documents that do not exist will be created in the corresponding folder of the target site. All derived contents will be marked as being in translation.

Table 6.28. Attributes of GetDerivedContents-Action

targetSiteIdVariable	
Required	yes
Description	The name of the variable that contains the id of the target site
masterContentObjects	
Required	yes
Description	The name of the variable that contains the list of content objects in the master site
derivedContentsVariable	
Required	no
Description	The name of the variable into which a list of all derived contents is stored
createdContentsVariable	
Required	no
Description	The name of the variable into which a list of all newly created contents is stored. If the workflow is subsequently aborted, these contents can be deleted by the action described in Section “RollbackTranslationAction” [368]

Example 6.27. Usage of GetDerivedContents-Action

```
<Variable name="siteId" type="String"/>
<AggregationVariable name="masterContentObjects" type="Resource"/>
<AggregationVariable name="derivedContents" type="Resource"/>
<AggregationVariable name="createdContents" type="Resource"/>
...
<AutomatedTask name="GetDerivedContents" successor="FollowUpAction">
```

```
<Action
class="com.coremedia.translate.workflow.GetDerivedContentsAction"
masterContentObjects="masterContentObjects"
derivedContentsVariable="derivedContents"
createdContentsVariable="createdContents"
targetSiteIdVariable="siteId"/>
</AutomatedTask>
```

GetSiteManagerGroupAction

This action is used to determine the user group that is responsible for managing the site. The name of this group is defined in the property `siteManagerGroup` of every site indicator. As this property is not required, the group administratoren will be used per default.

Table 6.29. Attributes of GetSiteManager-GroupAction

siteVariable	
Required	yes
Description	The name of the variable that contains the id of the site
siteManagerGroupVariable	
Required	no
Description	The name of the variable into which the site manager group is stored

```
<Variable name="siteId" type="String"/>
<Variable name="siteManagerGroup" type="Group"/>
...

<AutomatedTask name="GetTargetSiteManagerGroup"
successor="FollowUpAction">
  <Action
class="com.coremedia.translate.workflow.GetSiteManagerGroupAction"
siteVariable="siteId"
siteManagerGroupVariable="siteManagerGroup"/>
</AutomatedTask>
```

Example 6.28. Usage of GetSiteManager-GroupAction

ExtractPerformerAction

To perform an `AutomatedTask` with the same performer used in a previous `UserTask`, you can store the performer of the `UserTask` to the given workflow variable.

Table 6.30. Attributes of ExtractPerformerAction

performerVariable	
Required	no

Description	The name of the variable into which the performer of the current user task is stored
-------------	--

```
<Variable name="performer" type="User"/>
...

<UserTask name="Translate" successor="FollowUpAction">
  ...
  <EntryAction
class="com.coremedia.translate.workflow.ExtractPerformerAction"
    performerVariable="performer"/>
  ...
</UserTask>
```

Example 6.29. Usage of ExtractPerformerAction

CompleteTranslationAction

After successfully completing a translation workflow, the masterVersion of all translated contents will be set to the current version of their masters.

performerVariable	
Required	yes
Description	The name of the variable that contains the user in whose name this action performed. Typically, the user has been retrieved previously by the action described in Section “ExtractPerformerAction” [366] .
derivedContentsVariable	
Required	yes
Description	The name of the variable that contains all translated documents.
masterContentObjectsVariable	
Required	yes
Description	The name of the variable that contains all master content objects.

Table 6.31. Attributes of CompleteTranslationAction

```
<Variable name="performer" type="User"/>
<AggregationVariable name="targetContents" type="Resource"/>
...

<AutomatedTask name="Complete" successor="Finish">
  <Action
class="com.coremedia.translate.workflow.CompleteTranslationAction"
    derivedContentsVariable="derivedContents"
    masterContentObjectsVariable="masterContentObjects"
    performerVariable="performer"/>
  </AutomatedTask>
```

Example 6.30. Usage of CompleteTranslationAction

RollbackTranslationAction

If the master content is not needed in the target site, the translation workflow can be aborted with the `RollbackTranslationAction`. In this case all documents and folders that were created by the [Section “GetDerivedContentsAction” \[365\]](#) will be deleted. In addition, all target contents will be marked as no longer being in translation.

Table 6.32. Attributes of RollbackTranslation-Action

contentsVariable	
Required	yes
Description	The name of the variable that contains all documents and folders that have to be deleted during while rolling back the translation
derivedContents-Variable	no
masterContentObjectsVariable	no

		-noc
		tret
		-bo
		.sig
		-eD
		stef
		o t
		-st
		-ret
		-noc
		Out
		- b
		.sig

Example 6.31. Usage of RollbackTranslationAction

```
<AggregationVariable name="createdContents" type="Resource"/>
...
<AutomatedTask name="Rollback" successor="Finish">
  <Action
    class="com.coremedia.translate.workflow.RollbackTranslationAction"
    derivedContentsVariable="derivedContents"
    masterContentObjectsVariable="masterContentObjects"
    contentsVariable="createdContents"/>
  </AutomatedTask>
```

6.6.3 Deriving Sites

A predefined workflow exists to derive an entire site from an existing site. The derive-site workflow cannot be adapted and is available as a built-in workflow from the module `translate-workflow`. To upload the derive-site workflow, use **cm upload -n /com/coremedia/translate/workflow/derive-site.xml** on the command line.

Typically, the derive site workflow is started as a background process from the sites window of *CoreMedia Studio*. The workflow can be started by all members of the translation manager group, as configured in the property `translationManagerRole` of the `SiteModel` (see [section “Site Model” \[342\]](#)). After changing this property, you have to upload the workflow again, because uploading persists the current property value.

7. CoreMedia DXP 8 Brand Blueprint - Functionality for Websites

This chapter describes the *CoreMedia DXP 8 Brand Blueprint*

The content of [Chapter 6, CoreMedia DXP 8 e-Commerce Blueprint - Functionality for Websites \[244\]](#) also applies to *Brand Blueprint* unless noted differently.



7.1 Overview

The *Brand Blueprint* is an extension to the CoreMedia e-Commerce Blueprint. It provides a modern, appealing, highly visual website template that can be used to start a customization project. It demonstrates the capability to build localizable, multi-national, non-commerce web sites.

Based on a fully responsive, mobile-first design paradigm, the *Brand Blueprint* leverages the Twitter Bootstrap Grid and Design framework for easy customization and adaptation by frontend developers.

It scales from mobile via tablet to desktop viewport sizes and uses the CoreMedia Adaptive and Responsive Image Framework to dynamically deliver the right image sizes in the right aspect ratios and crops.

The responsive navigation visualizes 3 levels, even though the navigation structure can be arbitrary deeply nested. The floating header and the footer can be configured and re-ordered in content settings. Navigation nodes with URLs to external sites can be added via content.

Four new site-specific page types are introduced:

Brand Homepage

A long-scroller page with a "Super Hero" full screen teaser module at the top and visual "gap" headings that span the screen width. Several default placements demonstrate how layout settings can be done either directly on a placement or on a collection.

Brand Hero Page

A secondary navigation page with a large, emotional header image that spans the content width and several placements.

Corporate Detail Page

Corporate Detail Page: A simple, standard page with header, footer, main as well as above-main and below-main placements.

Corporate Detail Page with Sidebar

Corporate Detail Page with Sidebar: A variant of the Detail Page that displays a sidebar in a second column (desktop) or below the main placement (tablet, mobile). Useful for sections of a website with contact person teasers that can be inherited to detail article pages.

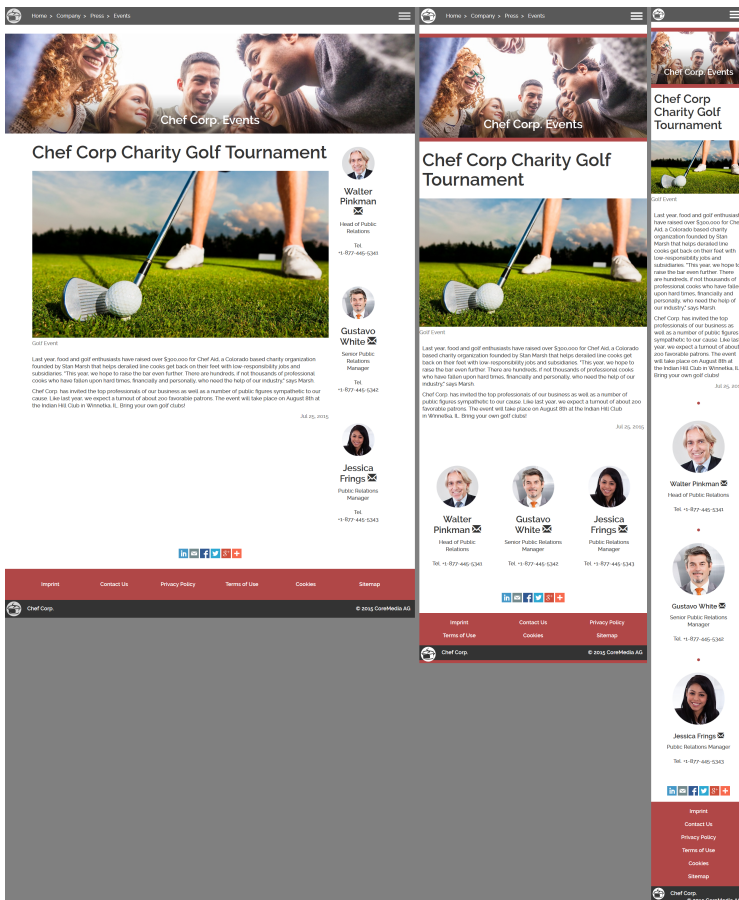


Figure 7.1. Corporate detail page for different devices

To simplify editorial use-cases for Brand casual users, a simplified page-centric editing model is supported via configuration:

- ➔ All centralized assets are stored in the "Assets" folder.
- ➔ Both navigation (Pages) and content are stored in the same repository folder structure that resembles the actual navigation structure of the site.
- ➔ The "Create page from Template" feature enables users to quickly create pages of the above layout variants. The new page is added to the navigation parent and adds a new default article and an image is cloned that can be replaced by right-clicking in the preview and by directly uploading a new image and pasting text from Word.
- ➔ Pages are now directly "teasable" and can be added as teasers to placements without the creation of a `Teaser` placeholder document.

- ➔ Any teaser can have a freely editable "Call-to-Action" button.

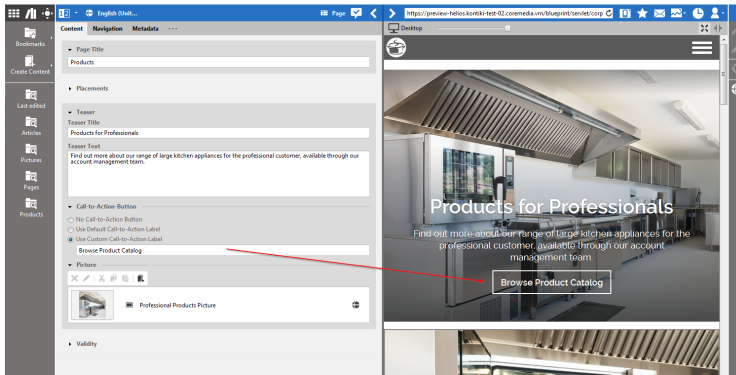


Figure 7.2. Teasable page with customized call-to-action button

Articles are also enhanced:

- ➔ Richtext with embedded images
- ➔ In-page carousel for multiple images with full-screen lightbox view
- ➔ Related content teasers
- ➔ They show a sidebar when defined in embedding page
- ➔ Configurable "Externally Visible Date" that remains unmodified upon re-publication

Several teaser types for different use cases:

Superhero	A large full-screen image, text and call-to-action button
Hero	A content-width spanning image, with text and call-to-action button
Default	Alternating left/right teaser module
Carousel	Touch-enabled carousel of teasers
Square	Square image teaser module
Claim/Claim (Circle)	A row with of three thumbnails and teaser texts, with optional round image CSS effect
Media List	Analogous to the Twitter Bootstrap style
Text	A text only teaser, for example, for press releases
Detail	Showing the actual content of an item, for example, to place a full article into a page

CMTeasable.ftl is the teaser template that will always be used as a fallback for any document type that has no specific override.



Figure 7.3. Different teasers on the Brand homepage



Multi-Language/Multi-Site features:

- ➔ Demo content in both English and German
- ➔ Language chooser in front-end that allows directly switching between language variations of the same content item

- All Studio translation, workflow and multi-site functionality is supported

SEO and conversion optimization:

- Call-To-Action Buttons
- Editable HTML Description Text
- Editable HTML Title Suffix (for example, " | Chef Corp.")
- Sitemap in both HTML and XML

Dynamic Content features:

- News and news list based on search-based lists
- Events and events list based on search-based lists

3rd Party Integrations:

- Google Universal Analytics integration with optimization feedback loop
- Optimizely Integration for A/B-Testing
- Full support for embedding dynamic third-party HTML/Javascript modules with examples for the following web services:
 - SurveyMonkey, Google Calendar, Google Forms, Pinterest, Twitter

Template creation

- Twitter Bootstrap is used as the layout foundation
- All frontend code is based on Freemarker templates
- CSS is built using the Sass and Grunt frameworks
- A simple Print.css is supplied
- Design and HTML were tested and optimized for Accessibility

Studio configurations:

- Configurable repository folder structure for Create dialogs
- Certain form elements can be site-specifically enabled or disabled through settings

CAE extensions:

- Shared template sets for teaser layout on both Placements and Collections via "Container" facade
- Specific `Preview.css` for Studio preview only

7.2 Website Features

This section describes general features of the *Brand Blueprint* website.

Long-scrolling pages

Brand Blueprint adds support for long-scrolling pages. Such pages usually have visual dividers to highlight different content sections - also called "Gaps" - that can have a parallax scrolling effect, that is, the picture inside the gap scrolls slower than the remainder of the page around it.

Long-scrolling pages with "Gap" dividers.

To configure gaps in the content, you have to add local settings.

Defining gaps for pages

For pages, you define placements for which the first item is rendered as a gap. Add the placement name to a Struct String List *placementsWithFirstItemAsHeader* in *Local Settings* or a *Linked Settings* content.

For example, on the Brand Homepage, "placement2" containing the "For Professionals" section is configured and consequently, the first item "Professionals Page" is not rendered as a regular teaser, but as a gap.

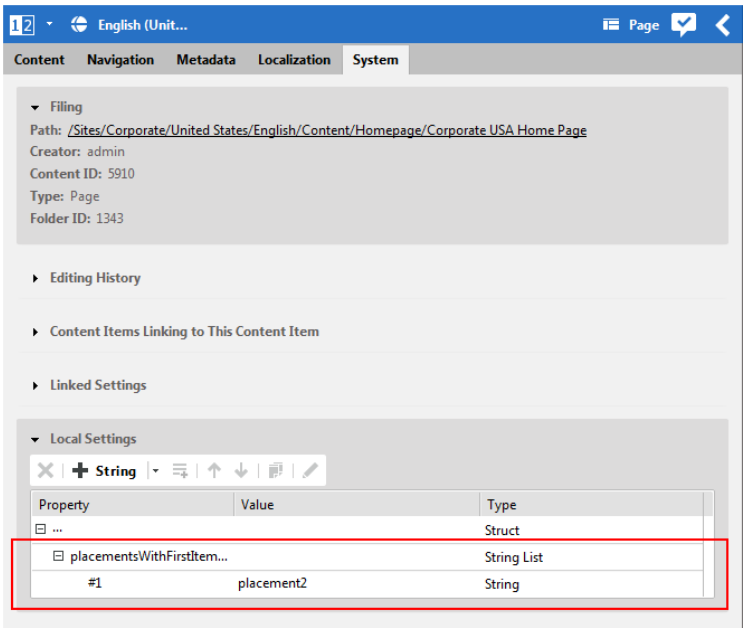


Figure 7.4. Define gaps for pages

Defining gaps for collections

For collections, you can define if the first item of the collection should be rendered as a gap. In *Local Settings* or a *Linked Settings* content, set a Boolean property *firstItemAsHeader* to true.

For example, in Placement 3 of the homepage, the first item is "For Consumers Collection" (Square). This collection has a settings document linked that sets the "firstItemAsHeader" to true.

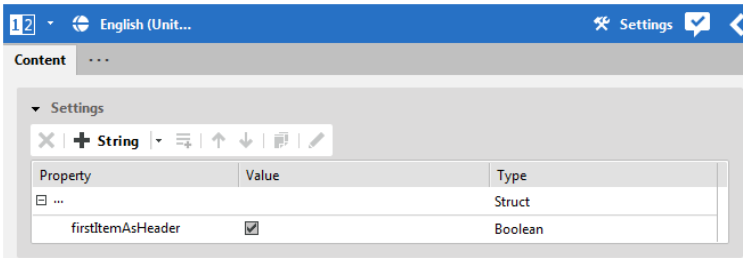


Figure 7.5. Setting content for collection with gap

Teaser rendering behavior for Placements and Collections

Brand Blueprint allows to set a *viewType* on a placement of a page grid. This can be used to ease editorial tasks - any item added to a pre-configured placement automatically gets the desired rendering *viewType* applied.

Teaser rendering behavior for placements and collections

Examples on the Homepage:

- ➔ The "Hero" placement is set to "Superhero", which automatically renders any added content item as a full-size Superhero teaser.
- ➔ The "Placement 1" is set to "Claim", which automatically renders any added content item as Claim teasers.
- ➔ The "Placement 3" is set to "Default", so that contained Collections can set their own *viewTypes* specifically.

These are default settings. You are free to customize the page grid to your needs.



Any setting on the placement in a page takes precedence over a setting on a collection that is inside the placement.

Example:

- ➔ A placement with *viewType*="Square" contains a Collection with *viewType*="Claim"
"Square" wins, collection items are rendered as "Square".

- A placement with `viewType="Default"` contains a Collection with `viewType="Claim"`

"Claim" wins, collection items are rendered as "Claim".

Additionally, through the Container facade, the same rendering templates also apply to Collections that are often used to define layout-specific rendering of teasers.

In the current Brand Blueprint release, it is not possible to set `viewTypes` directly on Teasers or other single content items. This can be achieved by creating additional `viewType` templates. Technically, the "Container" Facade only applies to lists of items, not to single items.

Collection view type lookup

In order to avoid duplication of layout templates for both Page Placements and Collections, a joint View Type Lookup is implemented via interface `com.coremedia.blueprint.common.layout.Container`. Just create a `CMViewtype` in `Options/Viewtypes/CMChannel/` for your Layout Variant, and it will be available for both Placements and Collections.

*Collection View Type
Lookup*

Configurable HTML Title Suffix

A site-specific HTML Title Suffix can be set by a technical editor via a Struct Setting.

For example, all page titles get appended " | Chef Corp.", and pages inside the Press Release page structure get appended " | News | Chef Corp."

`/Settings/Options/Bundles/Corporate_en`

- `customTitleSuffixText` (String)

*Configurable HTML
title suffix*

Editable HTML description meta tag

Each content item can now have a specific text editorially configured for the HTML Description Meta Tag via Studio.

For articles that are embedded directly on a page in Detail view, the HTML Description of the `CMChannel` is used, not the one of the `CMArticle`. For articles that are rendered in Detail view as "leaf" content in the context of another page, the HTML Description of the `CMArticle` is used.

*Editable HTML Descrip-
tion Meta Tag*



Configurable auto-shortening of teaser texts

As unedited teaser text is inherited directly from the detail body text, automatic shortening of too long teaser texts can be set for each teaser type with a site-specific Struct setting. The algorithm tries to shorten the text at word breaks.

Configurable Teaser text lengths

Options/Settings/CAEConfig

- `teaser.max.length` (Integer) - as in CM8
- `text.max.length` (Integer)
- `claim.max.length` (Integer)
- `hero.max.length` (Integer)
- `superhero.max.length` (Integer)
- `square.max.length` (Integer)

If the settings are not present, default values defined in templates will be used.

Configurable breadcrumb navigation

The breadcrumb display can be configured to omit first and last elements. For example, for a site you may want to suppress "Home" as a first breadcrumb element, or you might want to exclude "leaf content" (such as articles or other content items rendered in detail view) from the breadcrumb due to length and/or design constraints.

Breadcrumb display can be configured to omit first and last element

/Settings/Options/Bundles/Corporate_en

- `breadcrumbHideRootElement` (boolean)
- `breadcrumbHideLastElement` (boolean)

If the last navigation element is a page itself, it will be rendered in the breadcrumb, even if `breadcrumbHideLastElement` is set to "true".



Configurable Call-To-Action text for teasers

For each teasable content item, an editor can set a specific "Call-To-Action" (CTA) text that is displayed as a clickable button. The default text for the CTA is pre-configured per document type in the language Bundle document (for example,

Configurable Call-To-Action Text for teasers

Learn More for Articles). Users can override the default text or suppress the CTA button completely for each teaser.

The Studio forms for `CMTeasable` are extended for the Brand Blueprint to handle the CTA text. The CTA configuration is stored in the "localSettings" struct property with the following settings:

→ `callToActionCustomText` (String)

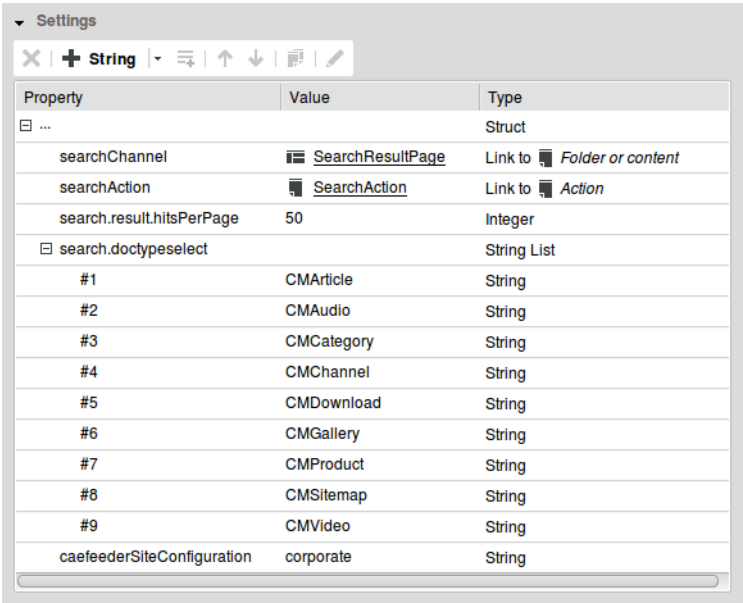
→ `callToActionDisabled` (Boolean)

7.3 Website Search

This section describes the configuration of the search functionality of the *Brand Blueprint* website.

Configuring search in content settings

Some aspects of website search are configurable in a site-specific Settings document. The site's root channel links to the Settings document *SearchConfiguration* with the settings used for that site.



Property	Value	Type
...		Struct
searchChannel	SearchResultPage	Link to Folder or content
searchAction	SearchAction	Link to Action
search.result.hitsPerPage	50	Integer
search.doctypeselect		String List
#1	CMArticle	String
#2	CMAudio	String
#3	CMCategory	String
#4	CMChannel	String
#5	CMDownload	String
#6	CMGallery	String
#7	CMProduct	String
#8	CMSitemap	String
#9	CMVideo	String
caefeedersiteconfiguration	corporate	String

Figure 7.6. SearchConfiguration Settings document

It contains the following settings:

Settings Property	Description
searchChannel	The channel used to render the search result page.
searchAction	Content of type CMAction with ID "search".
search.result.hitsPerPage	The number of hits shown on the search result page.
search.doctypeselect	The content types that appear in the search result. Subtypes must be listed explicitly.

Table 7.1. Brand website search settings

Settings Property	Description
caefeederSiteConfiguration	Contains the value <code>corporate</code> to select the <i>CAE Feeder Brand Blueprint</i> configuration for indexing content of the site. This enables page grid indexing as described in the next section.

Configuring page grid indexing

The *Brand Blueprint CAE Feeder* feeds CMChannel documents to the search engine so that pages can be found on the website. To this end, the CAE Feeder configuration specifies which parts of a page grid need to be indexed. This includes the configuration of relevant page grid sections, content types of linked contents and their properties.

Read section [Section 6.3.4, “Page Assembly” \[269\]](#) for an introduction to page grids.



The *Brand Blueprint CAE Feeder* is configured in the Spring bean definition file `component-corporate-caefeeder.xml` and its accompanied properties file `corporate-caefeeder.properties` in directory `src/main/resources/META-INF/coremedia` of the Blueprint module `modules/extensions/corporate/corporate-caefeeder-component`. The Spring XML file imports the content bean definitions and defines the following [FeedablePopulators](#) to index the page grid:

The `PageGridFeedablePopulator` takes properties from content linked in the page grid and adds them to the `textbody` index field when feeding a CMChannel. It is configured to feed the teaser properties of linked documents except for articles linked with view type "Detail" in which case the full article text is indexed with the channel. The `PageGridInlineContentFeedablePopulator` ensures that articles that are linked with view type "Detail" are not returned by the website search in addition to their page. To this end, it sets the index field `notsearchable` to `true` for such articles.

If a page grid placement contains a CMCollection document, then the contents linked in its `items` property are included as well - just as if they were linked directly in the page grid.

The mentioned `FeedablePopulators` are only used for documents if their site has a settings document that defines the setting `caefeederSiteConfiguration` with value `corporate`. This is the case for *Brand Blueprint* sites. The Spring application context file `component-corporate-caefeeder.xml` configures the site-specific activation of page grid feeding by adding the `FeedablePopulators` to the bean `siteSpecificFeedablePopulatorMap` for the value `corporate`.

The *Brand Blueprint* comes with a default configuration for indexing page grids of CMChannel documents. If needed, you can change the configuration in `component-corporate-caefeeder.xml` and `corporate-caefeeder.properties`. The following table describes the used Spring properties. All properties start with the prefix `corporate.search.pageGrid` which is abbreviated with `[c.s.p]` below.

Table 7.2. Page Grid Indexing Spring Properties

Property	Description
[c.s.p].contentType	The type of the contents with indexed page grid. Default: CMChannel
[c.s.p].name	The name of the struct property that contains the page grid. Default: placement
[c.s.p].excludedSections	Comma-separated list of ignored page grid sections. Default: header, footer, sidebar
[c.s.p].itemContentTypes	Comma-separated list of content types of considered page grid items. Contents of other types that are linked in the page grid are ignored and not indexed with the page grid. Default: CMChannel, CMArticle, CMTeaser, CMCollection, CMVideo, CMDownload, CMExternalLink, CMProduct
[c.s.p].itemTextProperties	The content properties of page grid items with a view type other than "Detail" that are indexed in the index field <code>textbody</code> of the page. This property takes a space separated string of document type properties. For each configured document type, the name of the type followed by an equal sign and a comma-separated list of property names is given. The configuration for the most specific document type of an item decides which item properties are used. The property lists are not merged with configurations for super types. This makes it possible to ignore properties in subtypes. Default: CMTeasable=teaserTitle,teaserText CMProduct=productName,shortDescription
[c.s.p].itemValidFromProperty [c.s.p].itemValidToProperty	The name of the date properties for visibility as described in Section 6.3.17, "Content Visibility" [301] . Content that is not currently visible is not indexed with the page. The <i>CAE Feeder</i> automatically reindexes after visibility has changed. Default: validFrom / validTo
[c.s.p].inlineContentTypes	Comma-separated list of content types used in the page grid with view type "Detail" for which the text properties

Property	Description
	are indexed with the page grid instead of the teaser properties. Default: <code>CMArticle</code>
<code>[c.s.p].inlineContentViewType</code>	The technical name of the "Detail" view type. Default: <code>full-details</code>
<code>[c.s.p].inlineContentTextProperties</code>	The content properties of page grid items with view type "Detail" that are indexed in the index field <code>textbody</code> of the page. This property takes a space separated string of document type properties. For each configured document type, the name of the type followed by an equal sign and a comma-separated list of property names is given. The configuration for the most specific document type of an item decides which item properties are used. The property lists are not merged with configurations for super types. This makes it possible to ignore properties in subtypes. Default: <code>CMArticle=title,detailText</code>
<code>[c.s.p].collectionContentType</code>	The content type of collection documents used in the page grid. Default: <code>CMCollection</code>
<code>[c.s.p].collectionItemsProperty</code>	The link property of collection documents to get the items of a collection. Default: <code>items</code>
<code>[c.s.p].collectionViewTypeProperty</code>	The link property of collection documents to get the view type for the items of a collection. Default: <code>viewtype</code>
<code>[c.s.p].configId</code>	An identifier that represents the configuration options. Default: <code>corporate</code>

Note that you must reindex from scratch with empty *CAE Feeder* database to apply the changes of the above configuration properties to all indexed documents. If it is okay to just apply the changes to newly indexed documents and if you don't reindex with empty *CAE Feeder* database, then you need to change the value of the `[c.s.p].configId` property to some other string constant, if you've changed one of the following properties (all starting with `[c.s.p].`): `name`, `excludedSections`, `itemContentTypes`, `itemValidFromProperty`, `itemValidToProperty`.



8. CoreMedia DXP 8 Editorial and Back-end Functionality

CoreMedia Digital Experience Platform 8 enhances *CoreMedia CMS* with additional functionality that is described in the following sections:

- [Section 8.1, “Studio Enhancements” \[387\]](#) describes extensions to *CoreMedia Studio* as the unified editing platform. The editorial usage of the features is described in the [Studio User Manual].
- [Section 8.2, “CAE Enhancements” \[414\]](#) describes extensions to the *Content Application Engine* the delivery module of *CoreMedia Digital Experience Platform 8*.
- [Section 8.3, “Elastic Social” \[418\]](#) describes extensions to *CoreMedia Elastic Social* that are integrated in *CoreMedia Digital Experience Platform 8*. The standard functionality of *Elastic Social* is described in the [Elastic Social Manual]
- [Section 8.4, “Adaptive Personalization” \[431\]](#) describes extensions to *CoreMedia Adaptive Personalization* that are integrated in *CoreMedia Digital Experience Platform 8*. The standard functionality of *Adaptive Personalization* is described in the [Adaptive Personalization Manual]
- [Section 8.5, “Third-Party Integration” \[440\]](#) describes the integration of third-party components, such as Optimizely, into *CoreMedia Digital Experience Platform 8*.
- [Section 8.6, “WebDAV Support” \[442\]](#) describes the standard integration of WebDAV to browse and create CMS content in the filesystem.

These modules are integrated into *CoreMedia DXP 8* and the example websites and add extended functionality to their default features.

8.1 Studio Enhancements

CoreMedia Blueprint enhances CoreMedia Studio with plugins for better usage. This ranges from improved content editors such as the image list editor, which shows a preview of a selected image, up to a complete taxonomy management.

- ➔ Image list editor, see [Section 8.1.1, “Image Link List Editor” \[387\]](#).
- ➔ Content chooser, see [Section 8.1.2, “Content Chooser” \[388\]](#).
- ➔ Document editors
 - Content query editor, see [Section 8.1.3, “Content Query Editor” \[390\]](#).
- ➔ Library, see [Section 8.1.6, “Library” \[394\]](#).
- ➔ Bookmarks, see [Section 8.1.7, “Bookmarks” \[395\]](#).
- ➔ External preview, see [Section 8.1.9, “External Preview” \[398\]](#).
- ➔ Content creation, see [Section 8.1.11, “Content Creation” \[400\]](#).
- ➔ Create content from template, see [Section 8.1.12, “Create from Template” \[405\]](#).
- ➔ Site selection, see [Section 8.1.14, “Site Selection” \[408\]](#).
- ➔ Upload dialog, see [Section 8.1.15, “Upload Files” \[408\]](#).

8.1.1 Image Link List Editor

The image link list editor (`<bp:imageLinkListPropertyField>`) is a simple extension to the standard link list editor. You can use it when you have a linklist that is primarily used to link images to a content item. It can show a thumbnail preview image of the linked content item holding the image. The image link list editor is able to deal with images in Articles, Collections, or related content within content items.

The actual thumbnail displayed for each linked content item depends on the type of the linked content item. The following rules apply:

Type	Image chosen
CMSelectionRules	Content item linked in the defaultContent linklist
CMCollection	First content item linked in the items linklist for which a selection rule applies
CMTeasable	First content item linked in the pictures linklist for which a selection rule applies

Table 8.1. Image Thumbnail selection rules

Type	Image chosen
CMPicture or CMImage	Image stored in the data property

Note that the rules above are applied in order, recursively, and each rule applies for more generic document types as well. For example, consider a situation where you have a `CMArticle` "A1" that has two content items linked in its `related` property, one `CMCollection` "C", and another `CMArticle` "A2". The collection in turn links to yet another two `CMArticles`, "A3" and "A4". For the first item linked in the article (a `CMCollection`), the respective rule applies that chooses the first item linked in the collection's `items` property, which is A2, a `CMArticle`. For this article, the rule for `CMTeasables` applies, since `CMArticle` inherits from `CMTeasable`. Therefore, the first item in the article's `pictures` property is inspected, which is a `CMPicture`. So ultimately, you will see thumbnails for

- The picture linked to from A3
- The picture linked to from A2

If you need to implement custom rules for thumbnail rendering for your own content types, you can do so by using the `ImageLinkListRenderer.registerRenderer(type_name, function)` method. See the API documentation of this class for details.

```
<bp:imageLinkListPropertyField propertyName="{PICTURE_PROPERTY_NAME}"
maxCardinality="{config.maxCardinality}" />
```

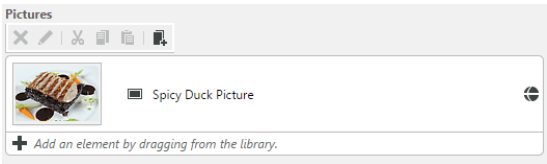


Figure 8.1. Image link list

The image link list editor of the image property of an article document.

8.1.2 Content Chooser

The content chooser allows the user to fill a link list by selecting documents from a list of checkboxes. The checkable items are documents read from a configurable folder and a configurable content type. The selection will be applied to the corresponding link list afterwards.

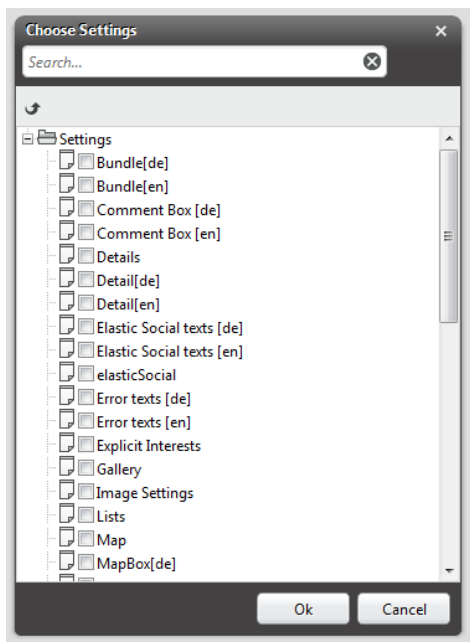


Figure 8.2. Content chooser

The content chooser is used for the settings property of the article document form.

How to configure a content chooser

Several usages of the content chooser can be found in the document forms of *Blueprint*. The `AdditionalToolBarItems` plugin is used to link the `OpenContentChooserAction` to the toolbar of linklist and provide a selection for it. The folder names where the selection items should be read from are passed to the action via the parameter `folders`. An example configuration of the content chooser action is shown below.

```
<ui:iconButton
tooltip="{Blueprint_properties.INSTANCE.Features_action_tooltip}">
  <baseAction>
    <contentchooser:openContentChooserAction

rootNodeName="{Blueprint_properties.INSTANCE.Features_root_name}"
dialogTitle="{Blueprint_properties.INSTANCE.Features_dialog_title}"

    iconCls="btn-linked-settings"
    bindTo="{config.bindTo}"
    valuesExpression="{config.bindTo.extendBy('properties',
'features')}"

folders="/Sites/Corporation-en/Editorial/Products/_Product Features"

    contentType="CMArticle"
```

```

        singleSelection="false"/>
    </baseAction>
</ui:iconButton>

```

In this example all `CMArticle` documents of the folder `/Sites/Corporation-en/Editorial/Products/_Product Features` are shown in the content chooser. Additional folder names can be set, using comma separated values.

8.1.3 Content Query Editor

Rather than having to maintain a collection of content items manually, you might want to just specify a search rule that updates a list of content items dynamically as new content gets added to the system. The content query editor provides a convenient interface to edit such rules.

For example, you can specify a rule that finds the latest five articles from your site's sports subsection, and displays them on a "latest sports news" section of your site's front page.

In the standard configuration of *Blueprint*, you can use the query editor to filter for content items according to the following aspects:

- the content item's document type
- the channel the content item belongs to
- the content item's modification date
- whether the content item is tagged with a given location or subject taxonomy

Furthermore, you can order the result set by different criteria, and you can specify a maximum number of hits in order to ensure proper layout on a column-based page design, for example.

Support for dynamic content queries is bundled in the *Studio* plugin, and the main component to use is `ContentQueryEditor.xml`. You can use the editor as shown in the following example.

```

<dcqe:contentQueryEditor bindTo="{config.bindTo}"
    queryPropertyName="localSettings"
    documentTypesPropertyName="documenttype"
    sortingPropertyName="order">
  <dcqe:conditions>
    <dcqe:modificationDateConditionEditor bindTo="{config.bindTo}"
      propertyName="freshness"
      group="attributes"
    />
  </dcqe:conditions>
  documentTypes="{['CMArticle', 'CMVideo', 'CMPProduct', 'CMPicture']}"
</dcqe:contentQueryEditor>

```

Example 8.1. Using the content query editor

```

                                sortable="true">
    <dcqe:timeSlots>
      <exml:object name="sameDay"
text="{QueryEditor_properties.INSTANCE.DCQE_text_modification_date_same_day}"
        expression="TODAY"/>
      <exml:object name="sevenDays"
text="{QueryEditor_properties.INSTANCE.DCQE_text_modification_date_seven_days}"
        expression="7 DAYS TO NOW"/>
      <exml:object name="thirtyDays"
text="{QueryEditor_properties.INSTANCE.DCQE_text_modification_date_thirty_days}"
        expression="30 DAYS TO NOW"/>
    </dcqe:timeSlots>
  </dcqe:modificationDateConditionEditor>
  ...
</dcqe:conditions>
</dcqe:contentQueryEditor>

```

In the example, the editor is configured to allow only for a single condition (a content item's modification date). You may combine the existing condition editors - there are predefined conditions for context, date ranges, and taxonomy links - or even write your own condition editors by extending `ConditionEditorBase.as`. Each condition editor provides the user interface for editing the respective condition, and must persist the actual search query fragment in a string property that will be written to the respective struct property. Also, all condition editors support the configuration of a list of document types that this condition may apply to. See the API documentation for the package `com.coremedia.cms.studio.queryeditor.conditions` for details.

When rendering the result of a search query in your CAE application, you can use `SettingsStructToSearchQueryConverter.java` to convert the search component that the editor stores in the struct property to an actual search query. See `CMQueryListImpl.java` for an example.

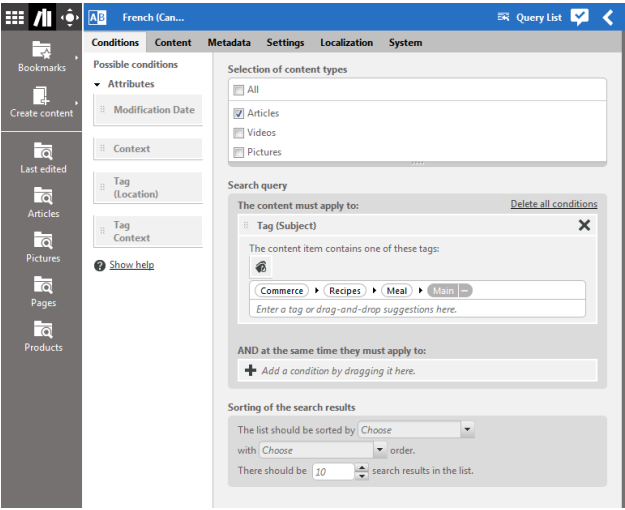


Figure 8.3. Content Query Editor

8.1.4 Call-to-Action Button

If you use teasers in your website, you want to animate the users to a specific action. To make this more explicit, *Brand Blueprint* renders a button on a teaser with a configurable text (see Figure 8.5, “Call-to-Action button in teaser view” [393]). By default, this text reads “Learn more”.

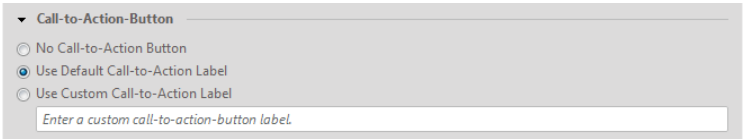


Figure 8.4. Call-to-Action-Button editor

You can either use the default text, define a content specific text or render no button.

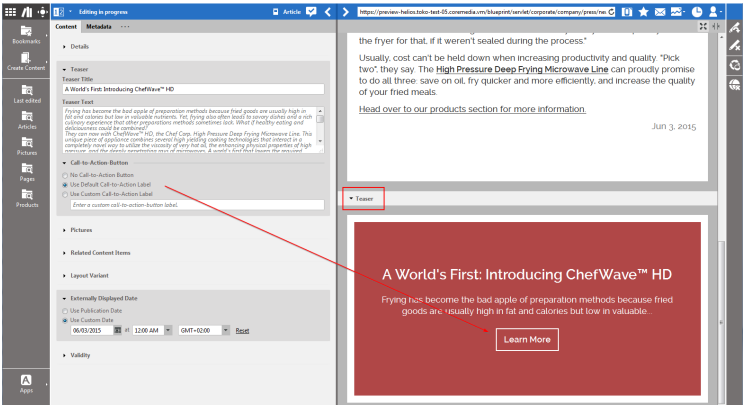


Figure 8.5. Call-to-Action button in teaser view

8.1.5 External Date

CoreMedia Corporate Blueprint feature



You may want to show a fixed publication date for a content, even when you change and republish this content later. To do so, *Studio* in *Brand Blueprint* contains an editor for an externally displayed date for all CMLinkable types:

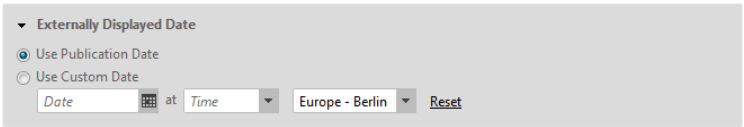


Figure 8.6. Externally displayed date editor

You can either choose that the publication date is used or that a fixed date is shown.

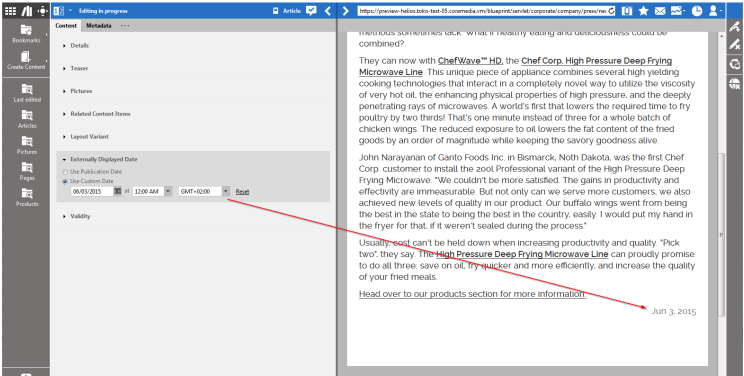


Figure 8.7. Setting an external date

8.1.6 Library

The library plugin uses the extension points of the *Studio* library to extend some basic features of it and to add some new ones.

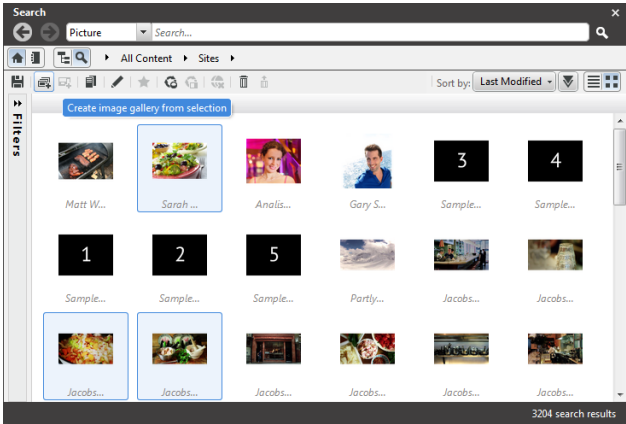


Figure 8.8. Image Gallery Creation Button

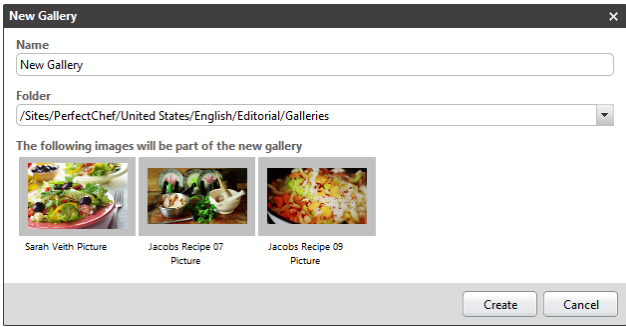


Figure 8.9. Image Gallery Creation Dialog

The image gallery creation dialog allows the user to create a new gallery document from an image selection. The images selected in the library are shown as thumbnails in the dialog when the 'Create Image Gallery' button is pressed. After the creation of the gallery, these images are automatically assigned to the list property of the document.

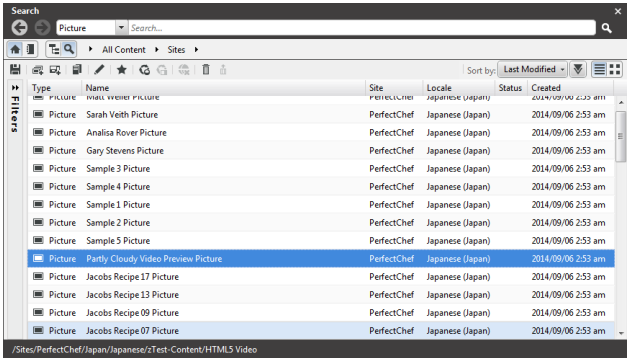


Figure 8.10. Library List View

The library plugin uses the library list view extension point to show some additional columns in the list view/search results. Additional columns are a site column, where the site name of a content item is displayed and a preview column, where images are shown as thumbnails. If the content item itself is not an image item, a referenced image is shown, such as the first picture of a gallery.

8.1.7 Bookmarks

The user can add and remove bookmarks using the bookmark action available on the preview toolbar, the library toolbar or the library list view's context menu.

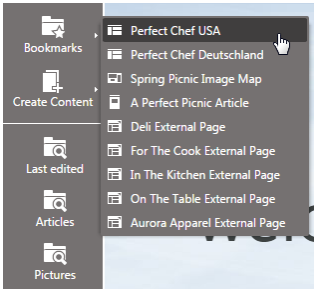


Figure 8.11. Bookmarks

8.1.8 External Library

Feature is only supported in *e-Commerce Blueprint*.



The external library previews data that is not located in the content repository. By using a separate REST extension, any data can be displayed by writing a provider class for it. Currently the external library supports RSS feeds and access to the video platform 'Kaltura'. The user can create new content using the external library

by using the *New Content* action. Depending on the data type, a new document is created and initialized with data of the selected library item.

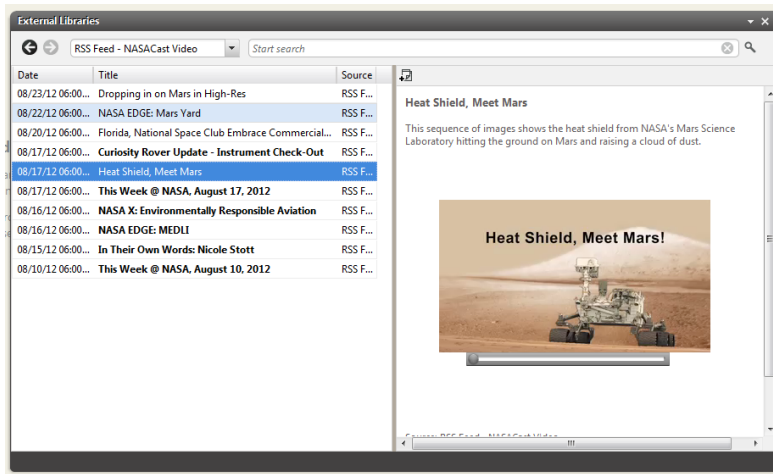


Figure 8.12. External library showing RSS feed items

How to configure the existing RSS source

The external sources that are available in the external library are configured via properties for each site in a settings document and in a global configuration settings document. The path information is configured for the class `ExternalLibraryResource` in the file `component-external-library-common-rest-extension.xml`. The name of the settings document that is located in these settings folders is `ExternalLibrary`.

To configure different source entries for a specific site, open the `ExternalLibrary` settings document using *CoreMedia Studio* and use the Struct editor to edit the configuration. Additional RSS sources can be added by cloning the corresponding `<Struct>` element and adept the URL of the feed. It is import that each configuration entry has a unique index value.

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <StructListProperty Name="externalLibraries">
    <Struct>
      <IntProperty Name="index">1</IntProperty>
      <StringProperty Name="name">RSS Feed - CNN.com - Top
Stories</StringProperty>
      <StringProperty
Name="dataUrl">http://rss.cnn.com/rss/edition.rss</StringProperty>
      <StringProperty
Name="providerId">rssProvider</StringProperty>
      <StringProperty Name="previewType">html</StringProperty>
      <StringProperty Name="contentType">CMArticle</StringProperty>

      <BooleanProperty Name="markAsRead">true</BooleanProperty>
    </Struct>
  </StructListProperty>
</Struct>
```

```
</StructListProperty>
</Struct>
```

For a detailed description about the elements and attributes see table below.

Table 8.2. Database Settings

Property	Description
index	The unique id entity of the entry as a numeric value.
name	The display name of the source, this name will be shown in the source combo box of the external library.
dataUrl	The data URL of the external source. It can be a HTTP URL or a database URL. It's up to the corresponding provider implementation to interpret this value.
providerId	The provider ID must match the Spring bean ID value
previewType	Describes the type of content to displayed, possible values are 'html' and 'video'. If required, the preview panel of the external library can be extended with additional view types.
contentType	The type of document that should be created when the "New document" button of the external library preview toolbar is pressed.
markAsRead	If true, the external library will remember if the user has read the entry.

How to implement an additional external data source

Additional data providers for the external library can be implemented using the development workspace extensions mechanism or using the existing workspace structure located in the module `external-library-rest-extension`. The following steps describe how to create and configure a new extension as a submodule of `external-library-rest-extension`.

- ➔ Open the `pom.xml` of the `external-library-rest-extension`
- ➔ In the module section, create a new module element with the name of the new extension, such as `sample-extension`
- ➔ Create the corresponding Maven submodule, ensure that the `pom.xml` file of the sample extension is configured the same way like the RSS or video extension's `pom.xml`.
- ➔ In the sample module create a new class that implements the interface `ExternalLibraryProvider`. Have a look on the existing provider implementation for help.

➔ Create the Spring configuration file `component-sample-extension.xml`

➔ Configure the provider in the XML file, for example like this:

```
<bean id="sampleProvider" class="
com.coremedia.blueprint.studio.externallibraryproviders.SampleProvider"
/>
```

➔ Add the new module as a dependency to the Studio web application module.

➔ Open the (site specific) settings content type `ExternalLibrary` in *Studio* and add the configuration entry for your library data provider. Ensure that the `providerId` value matches the bean ID of your provider class, in this case 'sampleProvider'. Use the preview type `html` and content type `CMArticle` for the configuration. It will fit most of your needs.

➔ Rebuild and restart the *Studio* web application and its dependencies.

8.1.9 External Preview

The external preview is a Studio utility that allows you to use one or more additional displays for Studio's preview based editing. When working with *CoreMedia Studio*, the external preview can be started by clicking on the 'open external preview' button that is located on the toolbar of the preview and following the instruction steps which are:

1. Insert the following URL in the address bar of the browser on your desktop computer or on your mobile device.

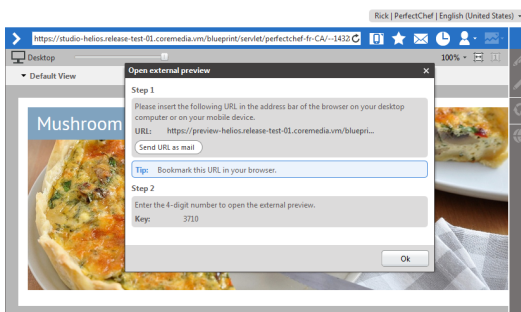


Figure 8.13. External Preview Dialog

2. Enter the 4-digit number to open the external preview. The code is mandatory to identify the corresponding user session and to prohibit monitoring the work of other users.

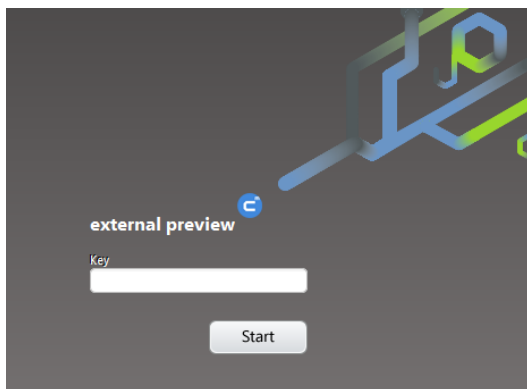


Figure 8.14. External Preview Login

The dialog shows the URL of the external preview. It can be invoked on any browser and device, including tablets to see how the document would look like on this device.

8.1.10 Settings for Studio

In order to use content-based settings not only for Content Application Engine usage but also for Studio, a new utility class `StudioConfigurationUtil` was introduced. Now you can, for example, configure paths used for the *Create Content* dialog (see [Section 8.1.11, “Content Creation” \[400\]](#)) in `CMSettings` content items.

The `StudioConfigurationUtil` class searches for bundles located at `<SITE_ROOT_FOLDER>/Options/Settings`, and falls back to `/Settings/Options/Settings` if no site-specific configuration bundle is found there. Bundle content items can be placed anywhere below these paths, and must be of type `CMSettings`.

You can use the `#getConfiguration(bundle, configuration, context)` method, where `bundle` is the name of the `CMSettings` document, and `configuration` is a path to a respective struct property. Optionally, you can also specify a `context`. The latter can be either a `Content` or a `Site`. If you provide `Content`, the site this content item belongs to is resolved, otherwise, the given site is used as the lookup context. If you omit the `context`, the current user's preferred site is used.

The utility class is fully dependency tracked, which means that you should wrap a `FunctionValueExpression` around returned values and bind the UI components that depend on the setting to this expression.

8.1.11 Content Creation

CoreMedia Blueprint provides additional buttons and actions to create new content besides the regular content creation action in the library. The user can click on the "Create content" menu on the favorites toolbar to open a selection of documents to create. The action is also available for link lists and several dialogs, like the 'Create' dialog of the external library.

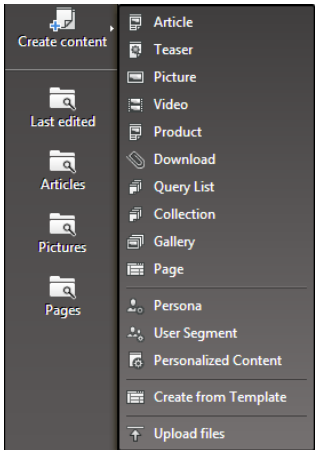


Figure 8.15. New content menu on the favorites toolbar

The user selects a content to create from the **Create content** menu of the favorites toolbar. Afterwards, a dialog opens where (at least) the document name and folder can be set.

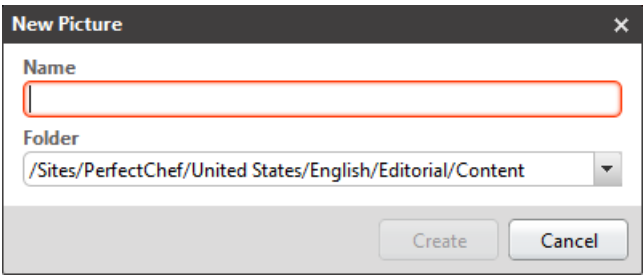


Figure 8.16. New content dialog

The user can decide if the content should be opened in a tab afterwards. The checkbox for this is enabled by default. The *Name* and *Folder* properties are the mandatory fields of the dialog. Depending on the content type the dialog shows different property editors, for example for *Page* content items, the additional field *Navigation Parent* is configured so that the user can select the navigation parent of the new page.

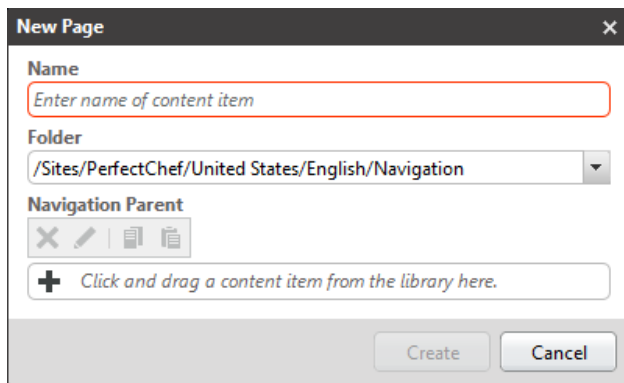


Figure 8.17. New content dialog for pages

The dialog can be extended in several ways and plugged into existing components using the predefined menu item or button components which will invoke the dialog. Also, the dialog provides a plugin mechanism for new property editors and allows you to customize the post-processing after the content creation, depending on the type of the created content. The following "How To" sections describe how to configure and customize the dialog.

How to add a 'New Content' menu item to the favorites toolbar

There are already some entries defined for this menu, most of them configured in the class `BlueprintFormsStudioPlugin.xml`. The menu can be extended using the `quickCreateMenuItem`:

```
<bp:newContentMenu>
  <plugins>
    <ui:addItemsPlugin>
    <ui:items>
    <bpb-components:quickCreateMenuItem contentType="MyDocumentType"/>
  ...
</bp:newContentMenu>
```

Separators can be added by:

```
<menuseparator cls="fav-menu-separator"/>
```

How to add a 'New Content' menu item to link list

There are two ways to add the content creation dialog to link lists. First is using the `QuickCreateToolbarButton` class and apply it to an existing link list using the `additionalToolBarItems` plugin. This will add one button to the toolbar of the link list to create a specific content type, for example creating a new child for the `CMChannel` document hierarchy:

```
<bp:extendedLinkListPropertyField bindTo="{config.bindTo}"
    propertyName="children">
  <bp:additionalToolBarItems>
    <tbseparator/>
    <bpb-components:quickCreateToolBarButton contentType="CMChannel"
  />
  </bp:additionalToolBarItems>
</bp:extendedLinkListPropertyField>
```



Example 8.2. Add content creation dialog to link list with quickCreateLinkListMenu

Figure 8.18. New content dialog as button on a link list toolbar

The second variant is that you apply a complete dropdown menu with several content types in it. By default, these content types are configured in the file `QuickCreateSettings.properties` that is part of the blueprint-base and overwritten with the file `NewContentSettingsStudioPlugin.properties` (see `BlueprintFormsStudioPlugin.xml`). The file contains a property `default_link_list_contentTypes` which contains the document types to display in a comma separated value format. This default can be overwritten by adding the `contentTypes` attribute to the `quickCreateLinklistMenu` element when the dropdown elements are declared in xml. The attribute value can have a comma separated format to support multiple content types too:

```
<bp:extendedLinkListPropertyField bindTo="{config.bindTo}"
    propertyName="header">
  <bp:additionalToolBarItems>
    <tbseparator/>
    <bpb-components:quickCreateLinklistMenu bindTo="{config.bindTo}"
        contentTypes="CMArticle,CMTeaser,..."
        propertyName="children" />
  </bp:additionalToolBarItems>
</bp:extendedLinkListPropertyField>
```

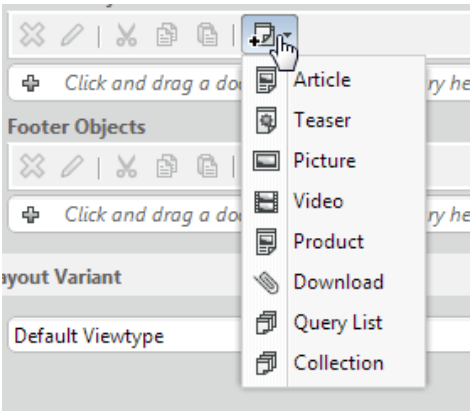


Figure 8.19. New content dialog menu on a link list toolbar

How to link new content to a link list

When the dialog is added to the toolbar of a link list by using the button component of the menu, the newly created content is automatically linked to the list. The dialog checks during the post-processing if the parameters *propertyName* and *bindTo* have been passed to it and will link the new content to the existing ones. The dialog always assumes that if these two parameters have been passed, the corresponding property is a link list, so using other properties with other types here will raise an error here.

How to add an event handler to the button or menu item

Both components, the `quickCreateLinkListMenu` and the `quickCreateToolBarButton` provide a configuration parameter called `onSuccess`. The method passed there will be executed after a successful content creation and must provide the signature:

```
method(content:Content, data:ProcessingData, callback:Function)
```

The `ProcessingData` instance "data" contains all the data entered by the user for the mandatory and optional properties of the dialog. The object is a `Bean` instance, so the values can be accessed by using `data.get(<KEY>)` calls. Since the new content dialog has already applied all dialog properties to the content, the retrieved new content instance already contains all inputted data.

Ensure that the callback handler is called once the post-processing is finished. Otherwise, the post-processing of the content can not terminate correctly and steps may be missing.



How to add a content property to the new content dialog

A new property editor that should be mapped to a standard content property can be defined in the file `NewContentSettingsStudioPlugin.properties`. The configuration entry supports a comma separated format in order to apply multiple property fields to the dialog. For example when the configuration entry `item_CMArticle=title,segment` is added to the properties file, each time the dialog is opened for a `CMArticle` document the String properties "title" and "segment" are editable in the dialog and will be applied to the new content.

Currently only text fields are supported, so do not configure a content property here that has a different format than "String".



How to add an event handler for a specific content type

The new content dialog allows you to apply a content type depending success handlers that are executed for every execution of the dialog. The success handler must implement the following signature:

```
method(content:Content, data:ProcessingData, callback:Function)
```

and is applied to the dialog by invoking:

```
QuickCreate.addSuccessHandler(<CONTENT_TYPE>, <METHOD>);
```

Unlike the `onSuccess` handler described in the previous section, these types of event handlers will be executed for every content creation of a specific type, no matter how and where the new content dialog is invoked from.



How to add a custom property to the new content dialog

Sometimes it is necessary to configure a value for the dialog that is not a content property. Instead, the value should be processed in the success handler. The dialog allows you to apply new editors to the dialog that are mapped to a specific field in the `ProcessingData` instance.

To apply a custom editor a corresponding factory method has to be implemented that will create the editor every time the dialog is created. This factory method is applied to the dialog then by invoking:

```
QuickCreate.addQuickCreateDialogProperty(<CONTENT_TYPE>,
    <CUSTOM_PROPERTY>,
    function (data:ProcessingData, properties:Object):Component {
        ...
        //for example return new CustomEditor(customEditor{properties});
    });
```

The `ProcessingData` instance is a bean, so it can be used to create `ValueExpressions` that are passed as parameters to the component. The predefined parameters are already applied to the `properties` object that is passed to the factory method. Additional properties can be added to this object, like the `emptyText` of an input field.

Make sure that the name of the custom property is unique and does not match an existing property of the given content type.



Since the new editor is shown for each dialog creation of the specific type, a success handler must be applied to the dialog too that processes the value:

```
QuickCreate.addSuccessHandler(<CONTENT_TYPE>,
    <myPostProcessingHandler>);
```

The processing handler must implement the same method signature like the ones defined for menu items or buttons:

```
method(content:Content, data:ProcessingData, callback:Function)
```

The custom property can be access in the handler by invoking:

```
data.get(<CUSTOM_PROPERTY>)
```

The post-processing of the dialog will execute the following steps:

1. create missing folders
2. create the new content
3. apply values to property fields (default processing)
4. invoke success handlers for custom processing (methods that have been applied through `QuickCreate.addSuccessHandler`)
5. invoke success handler configured for the button or menu items (methods that have been applied by declaring a value for the `onSuccess` attribute)
6. link content to a link list if parameters are defined
7. open created content
8. open additional content in background



Where do I find some examples?

Check the class `CMChannelExtension.as`. The class adds a `successHandler` for the creation of new `CMChannel` documents that is used to apply a value for the `title` property. Additionally the newly created `CMChannel` document is also linked to a parent (if available) that may have been provided by the `Navigation-LinkFieldWrapper` component that also has been added to the dialog.

8.1.12 Create from Template

As described in [Section 8.1.11, “Content Creation” \[400\]](#) when you create a Page content item in the *Create content* menu or from a link list, you will get a new and empty content item. If you want, on the other hand, create a Page with predefined content, or even a complete navigation hierarchy, you can use the **Create content** → **Create from Template** menu item. This will open a dialog where you can choose your Page from predefined templates.

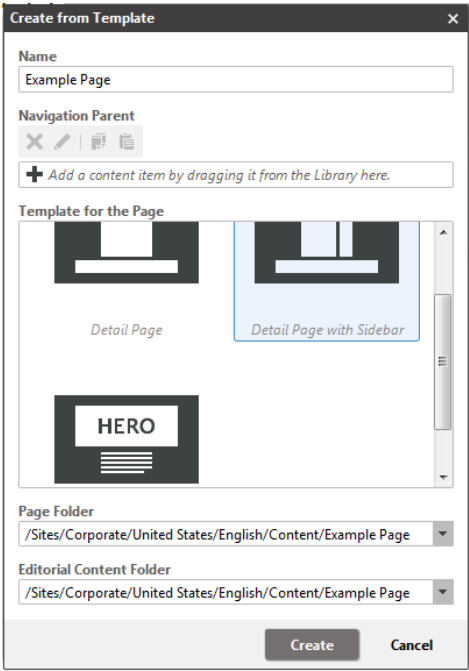


Figure 8.20. Create from template dialog

As with the standard **Create Page** dialog you can choose a name, the destination folder for the page and the navigation parent. The *Create from Template* dialog adds a template chooser from which you can select the template and a new combo box (*Editorial Content Folder*) where you can select a destination folder for the editorial content. The folder defined in the *Page Folder* combo box must not exist.

The suggested target paths for editorial content and content used to model the navigation are taken from a content-based setting from the bundle `Content Creation` (see [Section 8.1.10, “Settings for Studio” \[399\]](#) for an explanation of the content-based settings mechanism). You can modify the settings `paths.editorial` and `paths.navigation` to match your specific content tree.

Location of new template folders

By default, templates will be looked up in the following folders:

- ➔ Global: `/Settings/Options/Settings/Templates/CMChannel/`
- ➔ Site specific: `Options/Settings/Templates/CMChannel/`
- ➔ User's home folder: `{USER_HOME}/Templates/CMChannel/`

The lookup path is configurable in the *Studio* properties file `CreateFromTemplateStudioPluginSettings.properties` by changing the property `pagegrid_template_paths`. Additional entries can be added in a comma separated format.

Keep care when you configure a template path outside the site hierarchy or when you use the global templates location. It is possible that the preconfigured layout of a global template may not be available for the active site. Therefore, the page grid extending mechanism won't work anymore, since the page grid editor can't find the layout definitions of other pages.



How to add a new template folder

Template folders must have a specific format to be detected as template folders. Each template is defined in a separate folder inside the `Templates/CMChannel` folder. The folder must contain a `CMSymbol` document named "Descriptor" that might contain an additional icon and description for the template. The icon is used as a preview in the template chooser and the description will be shown as the template name in the template chooser.

Descriptor content

Each template folder must contain exactly one page document at root level, otherwise the folder will be ignored. If the template consists of several pages, the sub pages should be placed within a subfolder of the template. Editorial content (Article, Images ...) that is contained in these folders and is linked by Page templates will be copied to the destination, defined in the *Create from Template* dialog.

If the name and the description should be internationalized, create an additional `Descriptor` document next to the original descriptor and append the locale to the document name, "Descriptor_de" for the German version, for instance.

Localization

8.1.13 Site-specific configuration of Document Forms

With the `SiteAwareVisibilityPlugin`, you can show or hide document form elements (for example, property fields) depending of the activation of a "feature" for a specific site.

The `SiteAwareVisibilityPlugin` takes a parameter called "feature", which is a name for the feature. You can group two or more plugins by giving them the same feature name.

If you configure any ExtJS Component to use this plugin, that component only becomes visible when this feature is configured to be active for the site that the current content belongs to.

By default, the configuration for features of a site is done in a `CMSettings` document, which has to be named `<SITE_ROOT_FOLDER>/Options/Settings/Studio Features`

This settings bundle consists of a `StringList` named "features" and contains the string values that in turn need to be configured as desired in the `SiteAwareVisibilityPlugin`.

Example Usage

This plugin is used in the demo content of the *Brand Blueprint*. It hides the property editors for the "Call-To-Action-Button" and "Externally Displayed Date" in the Perfect Chef site, because these properties are not used in these templates, but are only used in the *Brand Blueprint*.

8.1.14 Site Selection

Since *CoreMedia Blueprint* provides multisite editing, a default working site can be configured in the settings dialog. If you select from *Preferred Site* for example 'Chef Corp. - German (Germany)' and then create a new article, it will be moved to a folder like this `/Sites/Chef Corp./Germany/...`

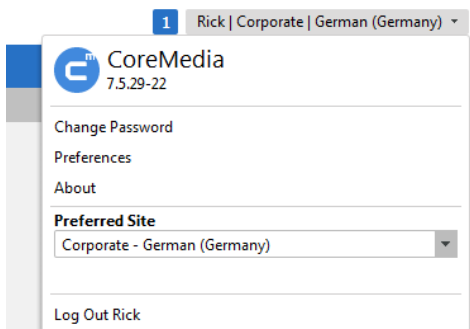


Figure 8.21. The site selector on the preference tab

8.1.15 Upload Files

The upload files dialog can be invoked from the new content menu or the library. The dialog shows a drop area and the folder combo box the uploaded documents will be imported into. Files can be dragged and dropped here from the desktop or the file system explorer. After the drop, the files are enlisted with a preview (if supported by the OS), a name text field and a mime type combo box. The mime type is automatically determined by the OS, but can be changed by selecting another value. After pressing the confirmation button the files are uploaded and corresponding documents are created. The user may choose to open the documents automatically after the upload is finished. Otherwise, the generated documents are checked-in.

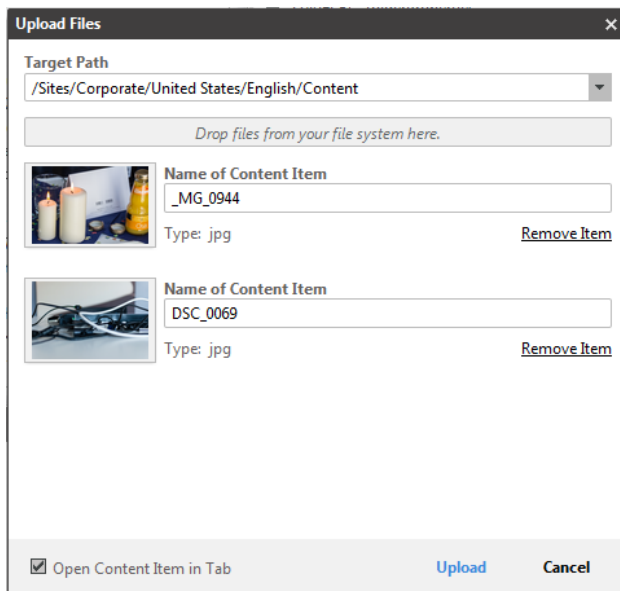


Figure 8.22. The upload files dialog

How to configure the upload settings

The upload settings are stored in the settings document `UploadSettings` in folder `/Settings/Options/Settings`. The default configuration has the following format:

```
<Struct xmlns="http://www.coremedia.com/2008/struct"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <StringProperty Name="defaultFolder">Editorial</StringProperty>

  <StringProperty
Name="defaultContentType">CMDownload</StringProperty>
  <StringProperty
Name="defaultBlobPropertyName">data</StringProperty>
  <IntProperty Name="timeout">300000</IntProperty>
  <StringListProperty Name="mimeType">
    <String>application/octet-stream</String>
    ...more mime types...
  </StringListProperty>
  <StructProperty Name="mimeTypeMappings">
    <Struct>
      <StringProperty Name="image">CMPicture</StringProperty>
      <StringProperty
Name="application">CMDownload</StringProperty>
      <StringProperty Name="audio">CMAudio</StringProperty>
      <StringProperty Name="video">CMVideo</StringProperty>
      <StringProperty Name="text">CMDownload</StringProperty>
      <StringProperty Name="text/css">CMCSS</StringProperty>
      <StringProperty
Name="text/javascript">CMJavaScript</StringProperty>
      <StringProperty Name="text/html">CMHTML</StringProperty>
    </Struct>
  </StructProperty>
</Struct>
```

```
<StructProperty Name="mimeTypeToMarkupPropertyMappings">
  <Struct>
    <StringProperty Name="text/css">code</StringProperty>
    <StringProperty
Name="text/javascript">code</StringProperty>
    <StringProperty Name="text/html">data</StringProperty>
  </Struct>
</StructProperty>
</Struct>
```

For a detailed description about the elements and attributes see table below.

Table 8.3. Upload Settings

defaultFolder	
Format	String
Description	Defines the default folder that is selected in the folder combo box of the dialog. The value supports site specific relative folders.
defaultContentType	
Format	String
Description	The default content type to create if the mime type of a file has no corresponding mime type mapping.
defaultBlobPropertyName	
Format	String
Description	The default blob property name to which the file blob is written to.
mimeTypes	
Format	String List
Description	The available mime types for the mime type combo box.
mimeTypeMappings	
Format	Struct
Description	Depending on the mime type the content type to generate is mapped here. Here the primary type or the whole mime type can be specified.
mimeTypeToMarkupPropertyMappings	
Format	Struct
Description	Depending on the mime type the markup property name to which the file is written
timeout	
Format	Integer
Description	The timeout in milliseconds for uploads, default value is 300000.

How to intercept the content's properties before creation

There is an example of a Content Write Interceptor contained in the Upload REST extension:

```
<bean id="pictureUploadInterceptor"
class="com.coremedia.blueprint.studio.rest.intercept.PictureUploadInterceptor">
    <property name="type" value="CMPicture"/>
    <property name="imageProperty" value="data"/>
    <property name="blobTransformer" ref="blobTransformer"/>
    <property name="extractor" ref="imageDimensionsExtractor"/>
</bean>
```

It is a Content Write Interceptor for `CMPicture` content type which scales an uploaded image blob to a configurable max dimensions and writes the image dimensions to the width and height String property of the image document. See the [CoreMedia Studio Developer Manual] for Content Write Interceptor.

8.1.16 Studio Preview Slider

Introduction

CoreMedia Studio's preview features a `slider` tool. The slider tool was build to let the user choose between devices with different resolutions in order to let the preview perform a responsive transformation of the page in the preview window. This means, that the preview will show the page as if it was to be viewed on a device with a different resolution than a "conventional" desktop display (that is a mobile device for instance).



Figure 8.23. The slider of the Studio Preview

Configuration of preview CAE

In order to enable the responsive slider functionality, you have to enable the use of `metadata tags` within the JSP templates. These tags are used for communication between the CAE and *CoreMedia Studio* in order to exchange meta information about the previewed page. (See *CoreMedia Studio Developer Manual* for more details about metadata tags). The following listing illustrates the enabled setting within the file `cae-preview-webapp/src/main/webapp/WEB-INF/application.properties`:

```
metadata.enabled=true
```

Integration of metadata tags in Freemarker templates

The following list illustrates the use of metadata tags in the `Page.body.ftl` template.

```
<#ftl strip_whitespace=true>

<!-- responsive design slider information for studio -->
<#assign sliderMetadata={
    "cm_preferredWidth": 1281,
    "cm_responsiveDevices": {
<!-- list of the devices.
naming and icons see: BlueprintDeviceTypes.properties
the default icons are in studio-core, but you can define
your own style-classes in slider-icons.css.
-->
<!-- e.g. iphone4 -->
    "mobile_portrait": {
        "width": 320,
        "height": 480,
        "order": 1,
        "isDefault": true
    },
<!-- e.g. iphone4 -->
    "mobile_landscape": {
        "width": 480,
        "height": 320,
        "order": 2
    },
<!-- e.g. nexus7 -->
    "tablet_portrait": {
        "width": 600,
        "height": 800,
        "order": 3
    },
<!-- e.g. ipad -->
    "hybrid_app_portrait": {
        "width": 768,
        "height": 1024,
        "order": 4
    },
<!-- e.g. nexus7 -->
    "tablet_landscape": {
        "width": 960,
        "height": 540,
        "order": 5
    },
<!-- e.g. ipad -->
    "hybrid_app_landscape": {
        "width": 1024,
        "height": 768,
        "order": 6
    }
    }
}
/>
```

To introduce new devices with even different resolutions, simply extend the content of the file appropriately.

Configuration in Studio

The configuration in Studio has to be made in the appropriate bundle files. The following listing shows the content of the file `modules/studio/blueprint-components/src/main/joo/com/coremedia/blueprint/studio/BlueprintDeviceTypes.properties`.

```
Device_mobile_portrait_icon=mobile-portrait-icon
Device_mobile_landscape_icon=mobile-landscape-icon
Device_tablet_portrait_icon=tablet-portrait-icon
Device_tablet_landscape_icon=tablet-landscape-icon
Device_notebook_icon=notebook-icon
Device_desktop_icon=desktop-icon

Device_mobile_portrait_text=Mobile
Device_mobile_landscape_text=Mobile
Device_tablet_portrait_text=Tablet
Device_tablet_landscape_text=Tablet
Device_notebook_text=Notebook
Device_desktop_text=Desktop
```

The configuration, which is relatively straightforward, consists of two parts. The top part of the configuration deals with the appropriate icons, that will be displayed for the according device type in the slider. The bottom part defines the text, that will be shown next to the slider. This configuration can be extended to introduce new device types with new device icons. For configuring the device icons, perform the following step:

- ➔ Declare a new class for the configured icon name in the file `modules/studio/blueprint-components/src/main/resources/META-INF/resources/joo/resources/css/slider-icons.css`.

8.2 CAE Enhancements

This section describes enhancements of the *Content Application Engine*.

- [Section 8.2.1, “Using Dynamic Fragments in HTML Responses” \[414\]](#) describes how context dependent HTML snippets can easily be used in a *Content Application Engine* application.
- [Section 8.2.2, “Image Cropping in CAE” \[416\]](#) describes how you can use cropped images in the CAE.

8.2.1 Using Dynamic Fragments in HTML Responses

Basic concept

Fragments of responses generated by the *Content Application Engine* may depend on a context, for example session data or the time of day. If fragments of a response may not be valid for every request, and responses are cached by reverse proxies (like Varnish or a CDN), it's necessary to exclude those parts from the response and load them separately using techniques like AHAH / Ajax or ESI.

To load the fragments, a link scheme and a matching handler handling the bean's type are needed.

CAE Implementation

In order to support loading of fragments in a generic and almost transparent way, beans are wrapped in a (`com.coremedia.blueprint.cae.view.DynamicInclude`) bean when they are included in the view layer. Whether the bean is wrapped or not is decided using `Predicate<RenderNode>` implementations that are called with the current `RenderNode`. A `RenderNode` represents the current "self" object and the view it's supposed to be rendered in. If any of the available predicates evaluate to true, the bean and view is wrapped as described above.

```
public class DynamicPredicate implements DynamicIncludePredicate {
    //only use DynamicInclude if view matches.
    private static final String VIEW_NAME="myView";

    public boolean apply(RenderNode input) {
        if (input == null) {
            return false;
        } else if (input.getBean() instanceof MyBean
            && VIEW_NAME.equals(input.getView())) {
            return true;
        }
        return false;
    }
}
```

Example 8.3. Predicate Example

The predicate has to be added to a predefined Spring bean in order to be evaluated:

```
<customize:append id="addMyDynamicPredicates"
bean="dynamicIncludePredicates">
  <list>
    <bean id="myPredicate"
          class="DynamicPredicate"/>
  </list>
</customize:append>
```

Example 8.4. Predicate Customizer Example

Render fragment placeholder

After wrapping the bean, the `DynamicInclude` is then rendered by the *Content Application Engine*.

`DynamicInclude` beans are rendered just as other beans by the *Content Application Engine*. By default, the view `DynamicInclude.ftl` is used to render the beans. It will either add a placeholder DOM element that can be used to load the fragment using AHAH, or an `<esi:include>` tag, depending on whether there is a reverse proxy telling the CAE to do so using the `Surrogate-Capability` header. This is described in the [Edge Architecture Specification](#).

Links to dynamic fragments

In order to generate a link for either AHAH or ESI, a separate link scheme must be created for each bean type that should be included dynamically.

If the fragment depends on the context (for example, Cookies, session or the time of day), the link scheme must have the prefix `/dynamic/` (see `UriConstants$Prefixes`) so that a preconfigured interceptor will set all Cache headers necessary that downstream proxies never cache those fragments. Matching Apache and Varnish rewrite rules are provided by *CoreMedia Blueprint*.

```
@Link(type = MyBean.class,
      view = "fragment",
      uri = "/dynamicfragment/mybean")
public UriComponents buildFragmentLink(Cart cart,
UriTemplate uriPattern,
Map<String, Object> linkParameters,
HttpServletRequest request) {

    UriComponentsBuilder result =fromPath(uriPattern.toString());
    //parameter "targetView" needs to be added
    result.queryParam("targetView", linkParameters.get("targetView"));

    return result.build();
}
```

Example 8.5. Dynamic Include Link Scheme Example

Handling dynamic fragments

These links have to be handled by using a handler. The handler has to use the `RequestParam` "targetView" to be able to construct a `ModelAndView` matching the values as originally intended in the `include` including the original bean.

Example 8.6. Dynamic Include Handler Example

```
@RequestMapping(value="/dynamicfragment/{mybean}")
public ModelAndView handleFragmentRequest(
    @PathVariable("mybean") String mybean,
    @RequestParam(value = "targetView") String view) {

    Object myBean = resolve(mybean);

    //do not create Page, return bean directly (!)
    ModelAndView modelAndView = createModelWithView(myBean, view);
    return modelAndView;
}
```

8.2.2 Image Cropping in CAE

As described in the [CoreMedia Studio Developer Manual] in chapter [Enabling Image Cropping](#), there are predefined crops, which can be applied to image rendering in the CAE. *CoreMedia Blueprint* comes with nine predefined cropping definitions as shown in the PerfectChef and Aurora sites.

- portrait_ratio20x31
- portrait_ratio3x4
- portrait_ratio1x1
- landscape_ratio4x3
- landscape_ratio16x9
- landscape_ratio2x1
- landscape_ratio5x2
- landscape_ratio8x3
- landscape_ratio4x1

The necessary settings for the image will be set by *Studio* once you open the image in *Studio*. To render images correctly even if they were not imported through *Studio* but for example by the *Importer* or WebDAV, the CAE provides a default cropping configuration for those images, which don't have the settings explicitly set. You will find these default settings in

```
/modules/shared/image-transformation/src/main/resources/framework/spring/mediatransform.xml
```


In this file, there is a list of the transformations mentioned above. Please refer to the Javadoc of `com.coremedia.cap.transform.Transformation` for all configuration possibilities. New Spring bean definitions of this class will be automatically injected to the `TransformImageService` that is responsible for all variant definitions.

Site Specific Image Variants

The features requires template changes. Examples for this are currently not supported by the *CoreMedia Blueprint*.



For the CAE, the class `TransformImageService` is responsible for loading site specific cropping information. The feature can be enabled by changing/adding the Spring property `imageTransformation.dynamicVariants` to `true`.

The `TransformImageService` requires a lookup of the Struct that contains the information about the image variants. Therefore, it must be injected with an instance of `VariantsStructResolver` which resolves the global and site specific image variants. The implementation of this interface is part of the `shared` module `image-transformation`, since the lookup is content type specific and therefore can not be part of the core.

8.3 Elastic Social

Feature is only supported in *e-Commerce Blueprint*.



CoreMedia Elastic Social is integrated into *CoreMedia Blueprint*. It includes the following features:

→ Comments and Reviews

Comments and reviews are supported for any kind of editorial CMS content items, for example articles and products. It is possible to configure for a context if writing comments or reviews is enabled and if it is allowed for anonymous or registered users. A review includes 5-star ratings with title and text.

Elastic Social provides aggregations like "Most Commented" or "Top Reviewed" content in a defined time interval for a certain context.

→ User Profiles

User profiles can be created using a registration flow and can be managed in the CAE by the user or in the *Studio* plugin "User Management".

→ Moderation

In the moderation of *Elastic Social* comments, reviews and user profiles can be edited, approved or rejected. In case of rejecting, a preconfigured template-based email can be sent directly or be modified by the moderator before sending it. A prioritization for comments, reviews or user profiles can be set. For all items that have to be moderated, premoderation, post-moderation or no moderation can be configured.

→ Password Reset

Password reset is available for registered users who authenticate directly with *Elastic Social*.

→ User Management

The *Elastic Social* user management in *Studio* includes a search for community users. The user management allows editing, searching, approving, blocking, ignoring and deleting users.

→ All Contributions

In the *All Contributions* section in *Studio* a list of all comments and reviews can be displayed. The list can be filtered by user, type, status or search term. Selected comments/reviews can then be edited, remoderated and marked for later editorial use.

- ➔ Display custom information in *Studio*
Custom information about users, comments or reviews can easily be integrated into the *Studio* moderation and user management via extension points.
- ➔ Emails
An email for a specific event can be sent by implementing the corresponding listener. Email templates can be created and edited in *Studio*.

8.3.1 Configuring Elastic Social

This section describes the configuration of the *Elastic Social* plugin.

Context settings for Elastic Social are defined in the following contexts:

- ➔ Root channel: Application context settings can only be defined in the root channel and can not be overwritten
- ➔ Every Channel: Channel context settings can be defined in every channel and are inherited or can be overwritten by child channels

Root Channel

The following context settings are defined for the root channel and can not be overwritten:

tenant	
Type	String property
Description	The tenant
Example	elastic
Default Value	
Required	true
userModerationType	
Type	String Property
Description	Moderation type for users
Example	PRE_MODERATION, POST_MODERATION, NONE
Default Value	NONE
Required	false

Table 8.4. Root Channel Context Settings

The context setting `tenant` is needed to define which tenant is used for a site.

Example 8.7. Root Channel Context Settings

```
<?xml version="1.0" encoding="UTF-8"?>
<Struct xmlns="http://www.coremedia.com/2008/struct"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <StructProperty Name="elasticSocial">
    <Struct>
      <StringProperty Name="tenant">
        elastic
      </StringProperty>
      <StringProperty Name="userModerationType">
        POST_MODERATION
      </StringProperty>
    </Struct>
  </StructProperty>
</Struct>
```

Every Channel

The following context settings can be defined per channel and are inherited or can be overwritten by child channels:

Table 8.5. Context Settings for Every Channel

Name	Type	Description	Example	Default value
enabled	Boolean Property	Enable/disable feed-back for the channel. If disabled, all other settings are ignored	true, false	false
commentType	String Property	Disable commenting generally by settings this property to DISABLED. Enable reading comments by setting this property to READONLY. Enable only registered users to write comments by settings the property to REGISTERED. Enable all users (registered and anonymous) to write comments by settings the property to ANONYMOUS. This property is only available if enabled is true.	DISABLED, READONLY, REGISTERED, ANONYMOUS	DISABLED
reviewType	String Property	Disable reviewing generally by settings	DISABLED, READONLY, RE-	DISABLED

Name	Type	Description	Example	Default value
		this property to DISABLED. Enable reading reviews by setting this property to READONLY. Enable only registered users to write reviews by settings the property to REGISTERED. Enable all users (registered and anonymous) to write reviews by settings the property to ANONYMOUS. This property is only available if enabled is true.	REGISTERED, ANONYMOUS	
commentModerationType	String Property	Moderation Type for comments.	PRE_MODERATION, POST_MODERATION, NONE	NONE
reviewModerationType	String Property	Moderation Type for reviews.	PRE_MODERATION, POST_MODERATION, NONE	NONE
filterCategories	LinkListProperty	Configures filter options for the comment moderation list. You can add navigation and taxonomy documents.		

Context Settings for Every Channel

```
<?xml version="1.0" encoding="UTF-8"?>
<Struct xmlns="http://www.coremedia.com/2008/struct"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <StructProperty Name="elasticSocial">
    <Struct>
      <BooleanProperty Name="enabled">
        true
      </BooleanProperty>
      <StringProperty Name="commentType">
        ANONYMOUS
      </StringProperty>
      <StringProperty Name="reviewType">
        REGISTERED
      </StringProperty>
      <StringProperty Name="commentModerationType">
        PRE_MODERATION
      </StringProperty>
    </Struct>
  </StructProperty>
</Struct>
```

Example 8.8. Context Settings for Every Channel

```

    </StringProperty>
    <StringProperty Name="reviewModerationType">
      PRE_MODERATION
    </StringProperty>
  </Struct>
</StructProperty>
</Struct>

```

8.3.2 Displaying Custom Information in Studio

You can show additional information inside the moderation tab and user management window of *CoreMedia Studio* by extending the *Studio* web application (server side) and modifying the `ElasticSocialStudioPlugin.xml` (client side).

Server Side: REST JsonCustomizer

Provide a `JsonCustomizer` to the *Studio* web application that adds the additional information to the data that is transferred from the REST back-end to the *Studio* app for users:

```

@Named
public class MyCommunityUserJsonCustomizer implements
JsonCustomizer<CommunityUser> {
    public void customize(CommunityUser communityUser, Map<String,
Object> serializedObject) {
        serializedObject.put("additional",
communityUser.getProperty("information", String.class));
    }
}

```

or for comments:

```

@Named
public class MyCommentJsonCustomizer implements
JsonCustomizer<Comment> {
    public void customize(Comment comment, Map<String, Object>
serializedObject) {
        serializedObject.put("additional",
comment.getProperty("information", String.class));
    }
}

```

Client Side (1): Display Custom Properties

Three extension points are provided for displaying custom properties for comments or users.

1. Extend the `commentExtensionTabPanel` to add components for comments that are displayed above the approve and reject buttons inside the moderation/archive tab (use `activeContributionAdministration` in the expression

for the `elasticPluginLabel` in order to reference the active contribution administration, depending on whether the moderation or the archive tab is active):

```
<ui:pluginRules>
...
<ui:rules>
...
<elastic:commentExtensionTabPanel>
  <plugins>
    <ui:addItemsPlugin>
      <ui:items>
        <panel title="additionalInformation" layout="form"
autoHeight="true" cls="elastic-extensionTab">
          <items>
            <es:elasticPluginLabel fieldLabel="additional"
expression="activeContributionAdministration.displayed.additional"/>
          </items>
        </ui:items>
      </ui:addItemsPlugin>
    </plugins>
  </elastic:commentExtensionTabPanel>
...
</ui:rules>
...
</ui:pluginRules>
```

2. Extend the `userProfileExtensionTabPanel` to add components for user profiles that are displayed above the approve and reject buttons inside the moderation tab:

```
<ui:pluginRules>
...
<ui:rules>
...
<elastic:userProfileExtensionTabPanel>
  <plugins>
    <ui:addItemsPlugin>
      <ui:items>
        <panel title="additionalInformation" layout="form"
autoHeight="true" cls="elastic-extensionTab">
          <items>
            <es:elasticPluginLabel fieldLabel="additional"
expression="contributionAdministration.displayed.additional"/>
          </items>
        </panel>
      </ui:items>
    </ui:addItemsPlugin>
  </plugins>
</elastic:userProfileExtensionTabPanel>
...
</ui:rules>
...
</ui:pluginRules>
```

3. Extend the `customUserInformationContainer` to add components that are displayed below the user meta information panel inside the user management view:

```

<ui:pluginRules>
  ...
  <ui:rules>
    ...
    <elastic:customUserInformationContainer>
      <plugins>
        <ui:addItemsPlugin>
          <ui:items>
            <container layout="form" autoHeight="true">
              <items>
                <es:elasticPluginLabel fieldLabel="additional"
expression="userAdministration.edited.additional"/>
              </items>
            </container>
          </ui:items>
        </ui:addItemsPlugin>
      </plugins>
    </elastic:customUserInformationContainer>
    ...
  </ui:rules>
  ...
</ui:pluginRules>

```

Client Side (2): Edit Custom Properties

For all three extensions points described above it is also possible to not just display but to edit/moderate custom properties. Instead of `elasticPluginLabel` just use `elasticPluginPropertyField`. This provides a text field for editing the property. Number or Boolean fields are not provided but can be constructed analogously. When you construct your own property field it is important to register the corresponding property as being moderated. This can either be done directly by your property field (c.f. `ElasticPluginPropertyFieldBase`) or you use the `registerModeratedPropertiesPlugin` for this purpose.

8.3.3 Adding Custom Filters for Moderation View

The list of moderated items of the *Moderation View* includes a filter section (see chapter *Using Elastic Social* of the *CoreMedia Studio User Manual*). By default, this section encompasses a filter for showing/hiding comments and users and for filtering comments in terms of comment categories.

It is possible to add further filters. You have to add your custom `filterFieldset` to the container `moderatedItemsSearchFilters` via the `addItemsPlugin`.

Each `filterFieldset` has to implement the method `buildQuery()`. For the case of moderation list filters, it has to return a string denoting comment/user properties and their desired values for filtering. Comment properties have to be prefixed with `"comments_"`. User properties have to be prefixed with `"users_"`.

For instance, if your filter returned `"comments_authorName=Nick"`, only comments written by an author named *Nick* would show up. You can combine multiple property-value pairs by separating them with `"&"`

Note that you probably have to provide appropriate indexes for your database in order to prevent your custom filters to have a negative effect on query performance.

8.3.4 Emailing

Emailing is supported by *Elastic Social* and can easily be incorporated for common use cases in a project. *Elastic Social* provides listeners which can be implemented to send emails (see *Elastic Social* documentation).

The `MailTemplateService` allows you to generate and send emails with a template name and parameters. The parameters define variables which can be used in the mail templates. Locale specific mail templates are used if a locale specific variant is available (locale specific suffixed name).

Per default all properties of a `CommunityUser` (the model for a user) are available as variable in a mail template. For example you can use `$givenName` to include the given name of a user (if you use *FreeMarker* for templating as *CoreMedia Blueprint* does). Additional parameters must be provided programmatically by passing them as map `additionalParameters` to the `MailTemplateService`.

In *CoreMedia Blueprint*, the following mail templates for the user and moderation processes are already provided with the example content. For each mail template, the template name and additional parameters are described.

If you want to use different additional parameters, redefine the variable in the mail template and pass the corresponding parameter in the `additionalParameters` map. All properties of the `CommunityUser` can be used in the templates without changing the code.

Table 8.6. Mail Templates

Use case	Template Name	Additional Parameters
Reset Password	passwordreset	<code>baseUrl</code> (reset password link)
Comment Replied	commentReplied	<code>replyText</code> , <code>replyAuthorName</code> , <code>replyDate</code> , <code>commentText</code> , <code>commentDate</code> , <code>commentUrl</code>

Mail Templates

8.3.5 Curated transfer

Contributions can be transformed into content objects for further use. In *CoreMedia Blueprint* the `curatedTransferExtensionPoint` must be configured to define the type of content:

```
<ui:pluginRules>
  ...
  <ui:rules>
    ...
    <elastic:curatedTransferExtensionPoint>
      <plugins>
        <ui:addItemsPlugin>
          <ui:items>
            <spacer width="10" height="100%"/>
            <tbseparator/>
            <button itemId="createArticleBtn" height="100%"
width="30">
              <baseAction>
                <ui:openDialogAction>
                  <ui:dialog>
                    <bp:newContentDialog folders="{['Editorial']}"
                                          skipInitializers="true"
                                          contentType="CMArticle"
onSuccess="{CuratedUtil.postCreateArticleFromComments}"
                                          openInTab="false">
                      </bp:newContentDialog>
                    </ui:dialog>
                  </ui:openDialogAction>
                </baseAction>
              </button>
            <tbseparator/>
          </ui:items>
        </ui:addItemsPlugin>
      </plugins>
    </elastic:curatedTransferExtensionPoint>
    ...
  </ui:rules>
  ...
</ui:pluginRules>
```

The content property can be configured in `CuratedTransferResource.java`:

```
private static final String CONTENT_PROPERTY_TO_COPY_TO =
"detailText";
```

8.3.6 Elastic Social Demo Data Generator

The *Elastic Social Demo Data Generator* is a standalone web application that generates the following entities:

→ Comments

- ➔ Reviews
- ➔ Users
- ➔ Blacklist entries

The generator simulates the online community of your website. You can, for example, simulate to have a comment written every 30 seconds and see how this affects the moderation list in the *Studio* plugin.

The *demodata-generator* web application is not started per default. To start the web application, start the `es-demodata-generator-webapp` or add a dependency to `es-demodata-generator-component` to your Preview CAE web application (Maven profile `preview-cae-with-es-demodata-generator`).

Once the web application is started, further control is provided either via a controller based JSP page or via manager based JMX access. Both ways provide functionality to start, stop and check the status of the generator. The JMX access enables fine grained configuration of the actual generation process.

In the *demodata-generator* web application, a demo data generator can be started simultaneously for each tenant.

The demo data generator initially generates a number of users (about 4000), images (about 30) and blacklist entries (about 20). Further data is generated incrementally depending on the comment generation frequency (`Interval` configuration property) and in relation to his value the frequency of other generated data like users, ratings, likes, etc (configuration properties ending with `Rate`).

Example 1: If the `Interval` property is 30, then a comment is generated every 30 seconds.

Example 2: If the `Interval` property is 30 and the `NewUserRate` property is 5, then a comment is generated every 30 seconds and a new user is generated each fifth iteration, that is every $5 * 30 = 150$ seconds.

Additionally, some rates depend on other rates.

Example 3: If the `NewUserRate` property is 5 and the `AnonymousUserRate` property is 4, then a new anonymous user is created only if a new user is generated in this iteration, that is each 20th iteration.

See [Table 8.8, “Elastic Social Demo Data Generator configuration” \[428\]](#) for a description of the rate properties which can be reconfigured via JMX.

JSP

Start the demo data generator for a tenant, such as 'corporate', with:

```
http://localhost:40088/es-demodata-generator-webapp/servlet/generate?tenant=corporate
```

To start the demo data generator for another tenant, change the tenant parameter. If no tenant is given, the default tenant is used.

The following parameters are available:

- `interval` (optional): Defines frequency of data generation in seconds. Default interval is 30 seconds. Effective only on startup.
- `stop` (optional): Stops the demo data generator for the given tenant.
- `tenant` (optional): Starts or stops the demo data generator for the given tenant. The tenant will be registered, if unknown. If omitted, the default tenant is used.

JMX

To manage the demo data generator via JMX for a specific tenant, navigate to the corresponding operation for the tenant and invoke it. The tenants are located in folder `com.coremedia.DemoDataGeneratorManager.blueprint`. If the JMX MBean is not available for a tenant, start the demo data generator via HTTP with the tenant parameter. The tenant will then be registered and the MBean becomes available.

The following operations are available via JMX:

Name	Description
start	Starts the demo data generator
stop	Stops the demo data generator
restart	Restarts the demo data generator
getStatus	Gets the status of the demo data generator
resetAllSettings	Resets all settings to default values

Table 8.7. Elastic Social Demo Data Generator operations

The following configuration is available via JMX:

Interval	
Description	Defines the frequency of demo data generation in seconds. A comment will be generated each interval.
Default	30

Table 8.8. Elastic Social Demo Data Generator configuration

AnonymousCommentRate	
Description	Defines the anonymous comment rate of newly created comments
Default	2
AnonymousLikeRate	
Description	Defines the anonymous like rate of newly created likes
Default	4
AnonymousRatingRate	
Description	Defines the anonymous rating rate of newly created ratings
Default	4
AnonymousUserRate	
Description	Defines the anonymous user rate of newly created users
Default	10
CommentComplaintRate	
Description	Defines the comment complaint rate
Default	50
CommentReplyRate	
Description	Defines the rate for comment replies
Default	5
CommentWithAttachmentsRate	
Description	Defines the rate for comment attachments of newly created comments
Default	5
LikeRate	
Description	Defines the new like rate
Default	2
NewUserRate	
Description	Defines the new user rate
Default	5
RatingRate	
Description	Defines the rating rate
Default	2
UserChangesRate	
Description	Defines the rate for changing users

Default	7
UserComplaintRate	
Description	Defines the rate of complaints about a user
Default	49
UserModerationType	
Description	Defines the moderation type used for user creation (<i>PRE_MODERATION</i> , <i>POST_MODERATION</i> or <i>NONE</i>)
Default	<i>POST_MODERATION</i>

Statistics are provided about the generated content and also about the number of available targets for comments that are available in your content repository. The following table only includes statistics which require parameters. For all other statistic data like the number of created comments just take a look at the available attributes when using the JConsole.

Name	Description
getNumberOfTeasablesForCommenting(String moderationType)	Returns the number of teasables with commenting enabled with the given moderation type or of all teasables if no moderation type is given (<i>PRE_MODERATION</i> , <i>POST_MODERATION</i> or <i>NONE</i>)
getNumberOfTeasablesForAnonymousCommenting(String moderationType)	Returns the number of teasables with commenting enabled for anonymous users with the given moderation type or of all teasables if no moderation type is given (<i>PRE_MODERATION</i> , <i>POST_MODERATION</i> or <i>NONE</i>)

Table 8.9. Elastic Social Demo Data Generator statistics

8.4 Adaptive Personalization

Feature is only supported in *e-Commerce Blueprint*.



CoreMedia Adaptive Personalization is integrated in *CoreMedia Blueprint*. It extends *CoreMedia Studio* and the CAE with the following features:

- Specific content types for personalized content, personalized search, user segments and test user profiles. See [Section 6.2.2, “Adaptive Personalization Content Types” \[253\]](#) for details.
- Specific editors in *CoreMedia Studio* for the content types.
- Different context sources to access taxonomy keywords, time related information and many more.
- Different search functions that can be used in personalized searches.

A main concept of *Adaptive Personalization* is context. When speaking about a context in terms of *CoreMedia Adaptive Personalization*, "a piece of data associated with a HTTP request" is meant. What data this is, is determined by the data sources you grant access to, for example a Geo Location service, your CRM or *CoreMedia Elastic Social*. *CoreMedia Adaptive Personalization* is a framework that manages these contexts and makes them available within your *CoreMedia* application. See the [Adaptive Personalization Manual] for detailed information, about how contexts work.

Context

In the file `personalization-context.xml` in module `p13n-cae` you can see which contexts are used in *CoreMedia Blueprint*.

The following sections describe details of the integration, the module structure, key integration points and some details on context.

- [Section 6.2.2, “Adaptive Personalization Content Types” \[253\]](#) gives an overview over the content types introduced by *Adaptive Personalization*.
- [Section 8.4.1, “Key Integration Points” \[432\]](#) describes key integration points of *Adaptive Personalization*
- [Section 8.4.2, “Adaptive Personalization Extension Modules” \[432\]](#) summarizes where to find *Adaptive Personalization* related source code in *CoreMedia Blueprint*.
- [Section 8.4.3, “CAE Integration” \[433\]](#) shows how *Adaptive Personalization* is embedded into the CAE.
- [Section 8.4.4, “Studio Integration” \[437\]](#) presents the *Studio* integration.

8.4.1 Key Integration Points

→ CoreMedia Elastic Social

As one example of providing context information that doesn't originate from the CMS, *CoreMedia Blueprint* comes with a ready-to-use integration for *CoreMedia Elastic Social*. As a result an editor can create Conditions in documents of type Personalized Search and User Segment that make use of a *CommunityUser*'s number of written comments, likes and ratings and/or simply information about the user himself (for example his given name).

Note that these features are only available when using *CoreMedia Adaptive Personalization* in combination with *CoreMedia Elastic Social*.

→ Taxonomies

As depicted in [Section 6.2.3, "Tagging and Taxonomies" \[254\]](#), each HTTP request against the CAE is augmented with Taxonomies. For example if a page with Content related to sport is shown, a "Sport" Taxonomy is associated with the request. *CoreMedia Blueprint* is configured to make these semantic classifications accessible to editors, that is, they can define Conditions on them in documents of type Personalized Search and User Segment.

8.4.2 Adaptive Personalization Extension Modules

CoreMedia Adaptive Personalization is integrated into the CAE using the CoreMedia project extension mechanism.

Adaptive Personalization Extensions

→ p13n

This is the basic *CoreMedia Adaptive Personalization* module. It provides essential implementation based on *Adaptive Personalization*, like definitions of contexts, custom content types and corresponding *ContentBeans*.

→ es-p13n

This extension combines *CoreMedia Elastic Social* with *CoreMedia Adaptive Personalization*. It offers *ContextSources* that publish properties of *Elastic Social*'s *CommunityUser* to the user's context collection so that it is available for *Adaptive Personalization*. This extension also offers convenience implementations for accessing and modifying user context, for example, the `com.coremedia.blueprint.personalization.elastic.InterestsService` which reads and writes contexts that refer to Subject Taxonomies.

See the [Blueprint Concept Guide] to learn more about Taxonomies.

➔ lc-p13n

This extension combines *CoreMedia LiveContext* with *CoreMedia Adaptive Personalization*. It consists of the CAE extension `lc-p13n-cae` and the Studio extension `lc-p13n-studio`.

Adaptive Personalization's Main Module in Detail

Module	Description	Content
p13n-cae	Generic CAE Plugin	ContentBean implementations for Adaptive Personalization content types, data view/Spring bean definitions, JMX configuration, ...
p13n-editor-lib	Runtime dependencies for <i>CoreMedia Site Manager</i>	XML schema for Adaptive Personalization's rule grammar, localization properties, ...
p13n-preview-cae	Preview CAE Plugin	Customizations specific to Preview CAE for example code to handle the evaluation of Test User Profiles
p13n-server	Bundles runtime dependencies for <i>Content Management Server</i>	XML schema for Adaptive Personalization's rule grammar, Adaptive Personalization content types, ... See the [Blueprint Concept Guide] for a description of the content types
p13n-studio	Studio plugin	DocumentForms corresponding to content types, custom UI components, localization properties, ...
p13n-studio-lib	Runtime dependencies for <i>CoreMedia Studio</i>	DocumentForms corresponding to content types, custom UI components, localization properties, ...
p13n-test-content	Encapsulates content and code for testing purposes	A prepared XML repository used during test execution, ...
p13n-uitesting	Wrappers for <i>Adaptive Personalization's Studio UI</i> components and the UI tests themselves	Wrapper for the rule property editor of a personalized content document, ...

Table 8.10. Adaptive Personalization's main Maven module in detail

8.4.3 CAE Integration

This section covers which contexts and `SearchFunctions` are available in *CoreMedia Blueprint*.

For a basic understanding of *Adaptive Personalization*'s key concepts and how to instrument them to fulfill project specific needs, please refer to the Adaptive Personalization Manual. To learn about CAE development in general see the [CAE Developer Manual].

Configured Contexts

To make use of *Adaptive Personalization* a CAE must be configured with contexts. In order to deliver personalized content these contexts will be analyzed at runtime each time a request is being processed.

CoreMedia Blueprint is shipped with the following contexts configured:

Context	Description
cookieSource_keyword	Cookie based ContextSource to track keywords associated with a Page.
cookieSource_subject_taxonomies	Cookie based ContextSource to track Subject Taxonomies See the [Blueprint Concept Guide] to learn more about Taxonomies.
cookieSource_location_taxonomies	Cookie based ContextSource to track Location Taxonomies See the [Blueprint Concept Guide] to learn more about Taxonomies.
referrerSource	Cookie based ContextSource to track the referrer URL of the first request of a session.
systemDateTimeSource	Provides access to time related information.
lastVisitedSource	Cookie based ContextSource to track a user's visited Pages.

Table 8.11. Adaptive Personalization contexts configured for CoreMedia Blueprint

Have a look at `personalization-context.xml` in module `p13n-cae` to see what kind of data is contained in the contexts. Especially, notice the used `ContextCoDec` implementations.

Refer to the [Blueprint Concept Guide] to learn about contexts in general and the [Adaptive Personalization Manual] to see how to implement a `ContextSource`.

Configured SearchFunctions

CoreMedia Blueprint comes with a content type called Personalized Search that represents a parametrized search query. You can use `SearchFunctions` to enrich the query String, which will be evaluated at request processing time. After evalu-

ation, the `SearchFunctions` are replaced with values from contexts resulting in a personalized search query.

CoreMedia Blueprint is shipped with the following `SearchFunctions` configured:

Table 8.12. Predefined `SearchFunctions` in *CoreMedia Blueprint*

Search Function	Description
<code>contextProperty</code>	<p>A search function that adds the value of a single context property to a search string.</p> <p>You can use the following parameters:</p> <ul style="list-style-type: none">→ <code>property</code> - the property of the context. Should be in the form: <code><context>.<property></code>→ <code>field</code> - the search engine field in which you want to search. <p>Example:</p> <p>The context named "bar" contains a property "foo" which has a value "42". Then, the search function <code>contextProperty(property:bar.foo, field:field)</code> will be evaluated to 'field:42'. That is, the <i>Search Engine</i> searches in the field named "field" for the value "42".</p>
<code>userKeywords</code>	<p>A search function that selects from a user's context a set of keys that fulfill a weight constraint.</p> <p>You can use the following parameters:</p> <ul style="list-style-type: none">→ <code>limit</code> - Limits the number of returned keys (a negative or missing value means no limit)→ <code>field</code> - The search engine field which should be searched→ <code>threshold</code> - The minimum weight of keys to be returned→ <code>context</code> - The context containing the keys <p>Example:</p> <p>The context object named <code>myContext</code> contains the properties (<code>foo, 0.8</code>), (<code>bar, 0.5</code>), (<code>zork, 0.1</code>). Then, the search function <code>userKeywords(threshold:0.5, limit:-1, field:field, context:myContext)</code> will be evaluated to <code>field:(foo bar)</code> and the search function <code>userKeywords(threshold:0.5, limit:1, field:field, context:myContext)</code> will be evaluated to <code>field:(foo)</code>.</p>

Search Function	Description
userSegments	<p>A search function that selects the set of user segments the active user belongs to.</p> <p>You can use the following parameters:</p> <ul style="list-style-type: none">→ <code>field</code> - The search engine field which should be searched→ <code>context</code> - The context that contains segment properties <p>Example:</p> <p>The context object named "myContext" contains the properties ('content:42', true), ('content:44', false), ('content:46', true). Then, the search function <code>userSegments(field:field, context:myContext)</code> will be evaluated to "field:(42 46)". This function is intended to be used with the user segmentation feature of <i>CoreMedia Adaptive Personalization</i>, which uses property keys of the form <code>content:<segmentId></code> (where <i>segmentId</i> is the numeric content id of a user segment) to represent segments in a user's context.</p>

See the [Blueprint Concept Guide] and the [Adaptive Personalization Manual] for more information on `SearchFunctions` and the content type Personalized Search.

Enabling Test User Profiles in the Preview CAE

To make the Test Profile Selector work, the *Preview CAE* is provided a special context configuration: Its `ContextCollector` extends all properties of the generic CAE `ContextCollector`, but also adds a `TestContextSource` (see `p13n-preview-cae-context.xml` in `p13n-preview-cae`). This `TestContextSource` makes contexts from Test User Profile documents available by extracting the following information from a Test User Profile:

- Arbitrary contexts held in a plaintext blob using the `PropertiesTestContextExtractor`
- Subject/Location Taxonomies determined in a Struct property using the `TaxonomyExtractor`

The activation of the `TestContextSource` is triggered by passing a special URL parameter - `TestContextSource.QUERY_PARAMETER_TESTCONTEXTID` - to the *Preview CAE*.

Further reading:

- ➔ See [Section “Using Personas” \[438\]](#) for the Test Profile Selector's usage
- ➔ Refer to the [Blueprint Concept Guide] to learn more about documents of type Test User Profiles and Personalized Content
- ➔ The [Adaptive Personalization Manual] explains how to specify contexts in a Test User Profile document

8.4.4 Studio Integration

This section covers which `Conditions` are configured in *CoreMedia Blueprint* and how to use the Test Profile Selector.

For a basic understanding of *Adaptive Personalization*'s key concepts and how to instrument them to fulfill project specific needs, please refer to the [Adaptive Personalization Manual]. To learn about *Studio* development in general see the [Studio Developer Manual].

Configured Conditions

To make use of contexts in documents of type Personalized Content or User Segment corresponding `Conditions` have to be implemented in Studio. *CoreMedia Blueprint* provides `Conditions` for all contexts listed in [Table 8.11, “Adaptive Personalization contexts configured for CoreMedia Blueprint” \[434\]](#).

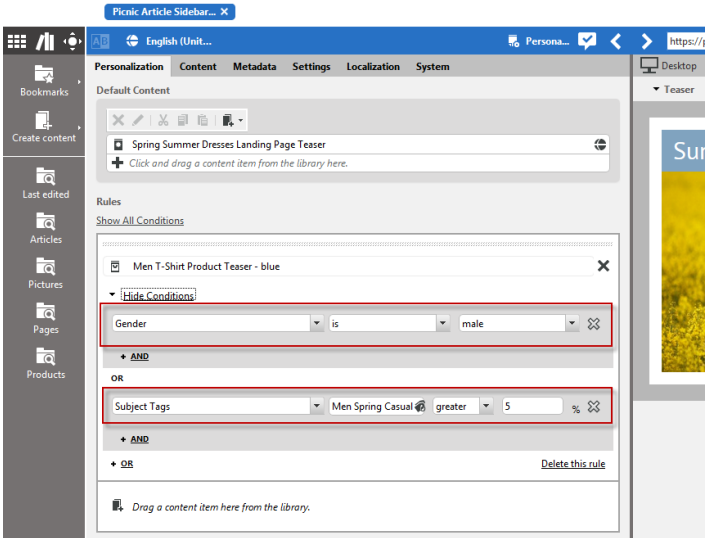


Figure 8.24. Conditions in Personalized Content and User Segment documents

Have a look at `CMSelectionRulesForm.exml` and `CMSegmentForm.exml` in module `p13n-studio` to get an idea about how to plug in `Conditions`.

Using Personas

A Persona is a collection of artificial context properties under the control of the editors. The type of properties to use depends on the configured contexts. For example the name of a visitor is a String while the number of likes performed is a numeric value.

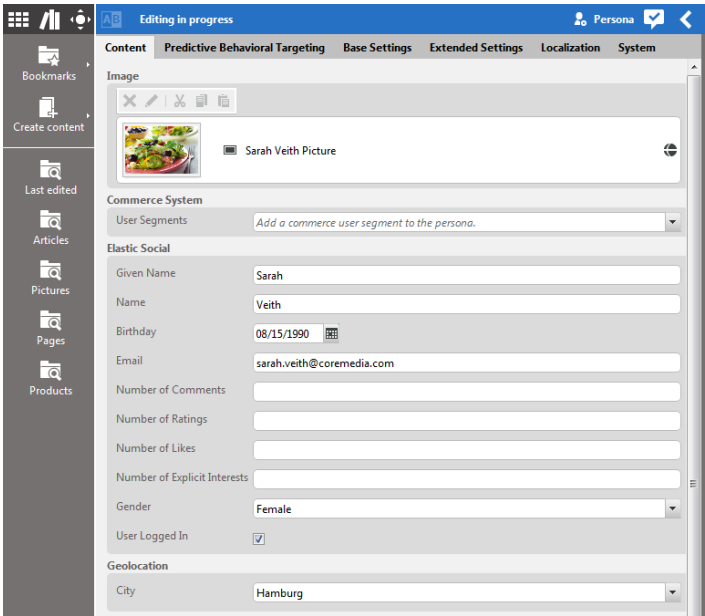


Figure 8.25. Defining artificial context properties using Personas

See [Table 8.11, “Adaptive Personalization contexts configured for CoreMedia Blueprint” \[434\]](#) for an overview of configured contexts.

Using the Persona Selector an editor is able to test a Personalized Content document. By choosing a specific Persona all its contexts are activated within the Preview CAE. As a result, the Preview CAE renders content as if corresponding contexts were available at request processing time.

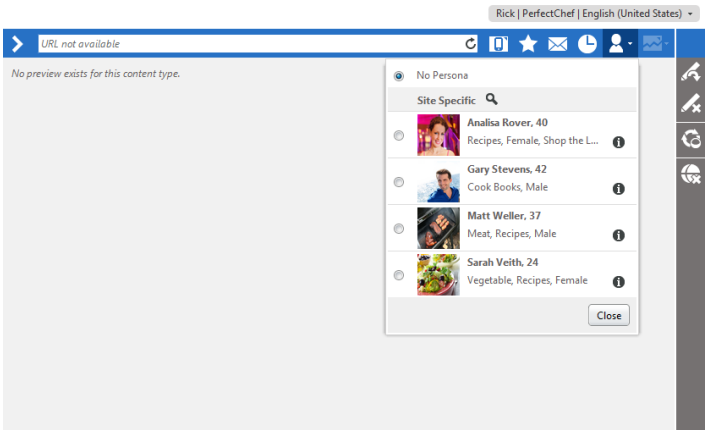


Figure 8.26. Selecting Personas to test Personalized Content and User Segment documents

See section [Section “Enabling Test User Profiles in the Preview CAE” \[436\]](#) to learn how Personas are integrated into the Preview CAE. The Adaptive Personalization Manual describes in detail how to create and use Personas.

8.5 Third-Party Integration

CoreMedia Blueprint comes with default integrations of third-party software.

8.5.1 Optimizely

Optimizely is a service which offers A/B testing for your website. You can define different rules for A/B testing using the editor on the Optimizely website.

CoreMedia Blueprint integrates Optimizely to perform A/B testing for channels. To enable this for a certain channel:

1. Open the target channel in *CoreMedia Studio*.
2. Open the Settings tab.
3. Create a struct property of type Boolean in the "Local Settings" section with the name `optimizely.enabled` and set it to true.
4. Create a struct property of type String in the "Local Settings" section with the name `optimizely.id.external.account` and enter the project id you got from Optimizely.

Now the JavaScript provided by Optimizely will be included in each site of this channel and you can measure your website improvements. The "Local Settings" will be inherited to sub channels. That means, the project id can be set, for example for the root channel once and you can disable Optimizely using `optimizely.enabled` for each sub channel individually.

Remember also, if you adjust target links on your website the target site must be available in the channel, too. Otherwise, the Optimizely script will not be included on the target site and clicks cannot be tracked.

8.5.2 Open Street Map Integration

The Open Street Map project creates and distributes free geographic data. *CoreMedia Blueprint* is prepared to include the project to display the location of location based taxonomies, but map integration are not included in the default templates.

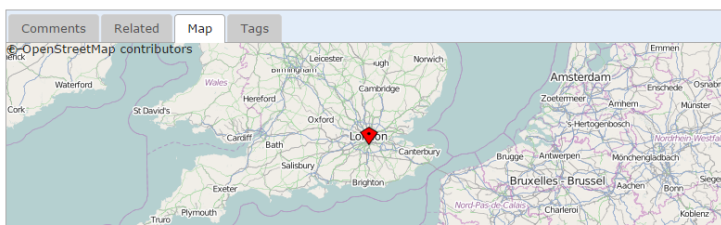


Figure 8.27. Example for a Open Street Map integration in a website

In order to use Open Street Map on your site, you have to create a settings document and link it to the root channel of your site. The JavaScript for Open Street Map will be loaded using an aspect that is only enabled if the corresponding settings property is set. The available settings for Open Street Map are shown in the table below and must be configured to enable the map in the CAE. A template renders a map segment according to geographic coordinates stored in the string property `latitudeLongitude` of a linked location content, and pinpoints the matching location with a marker image (see `CMTeasable.map.jsp` for a usage example).

Setting	Struct Type	Mandatory	Description
detail.show.map	Boolean Property	not	If true, the Open Street Map aspect will be enabled.
map.zoom	Int Property	no	The map zoom factor to use.

Table 8.13. Settings for Open Street Map Integration

8.5.3 Google Analytics Integration

Brand Blueprint feature



Brand Blueprint integrates Google Analytics into the website to get performance feedback. Have a look into the CoreMedia Operations Basics to learn about the configuration.

Section 4.7.5, “Getting Analytics Feedback” in *CoreMedia Studio User Manual* and Section 4.2.5, “Adding Site Performance Widget” in *CoreMedia Studio User Manual* describe how to get the performance feedback in *CoreMedia Studio*.

8.6 WebDAV Support

CoreMedia Blueprint provides WebDAV support both for browsing the content repository and to create or modify content in the `webdav.properties` file. It is possible to access the content repository by the WebDAV web application using a browser or a connection to a network share.

To connect to the WebDAV web application using a network drive on Windows you need a certificate issued for the WebDAV server, for example `*<.my-dns-hostname>` (such as `*.blueprint7.coremedia.com`). This certificate must be part of the trusted root certificates on the client Windows machine.

Using a network drive allows you to browse the repository and to simply create content from files on your computer by drag and drop. In case of Windows you can map a network drive to the WebDAV web application. Remember that Windows only accepts connections over HTTPS and that the server certificate is part of the trusted root certificates on the client machine.

CoreMedia Blueprint's WebDAV configuration chooses the type of the created content depending on the MIME type of the files sent to the WebDAV application. The default configured types are:

- `CMPicture` for images
- `CMAudio` for audio files
- `CMVideo` for video files
- `CMDownload` for downloads

Required properties like title will automatically be set by a `BlueprintWebDav FileSystemListener` which can be configured for the WebDAV web application.

8.7 Advanced Asset Management



The *CoreMedia Advanced Asset Management* module needs to be licensed. If the component is not licensed you are welcome to discover some functions of the component in CoreMedia Studio. Be aware that *CoreMedia Advanced Asset Management* is not working in the live system. If you are interested in this component, get in touch with your contact at CoreMedia.

If you want to remove the *CoreMedia LiveContext Product Asset Management* extension from your workspace follow the instructions described in [Section “Removing the Advanced Asset Management Extensions” \[151\]](#).

CoreMedia Advanced Asset Management consists of two parts:

- An Asset management component with new content types where you can manage your digital assets and licences.
- A connection with the IBM WCS system where you can display your assets in the IBM system.

CoreMedia Asset Management allows you to store and manage your digital assets (for example, high-resolution pictures of products) and corresponding licenses in the CoreMedia system. You can customize the storage of assets and the set of available asset types and rendition formats.

Managing Assets

A rendition is a derivative of the raw asset, suitable for use in output channels, possibly with some further automated processing. A rendition might be, for example, a cropped and contrast-adjusted image in a standardized file format whereas the original file might be stored in the proprietary format of the image editing software in use.

CoreMedia Asset Management integrates with Adobe Drive so you can, for instance, edit images stored in *CoreMedia DXP 8* directly in Photoshop.

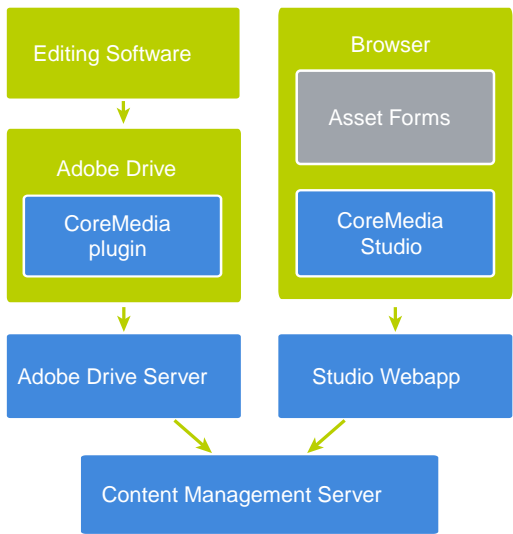


Figure 8.28. Overview over asset management part

From such assets, you can create common content items, such as `Picture` or `Download` which you can use to enrich products and product variants (products for short) in the *IBM WCS*.

Enhancing *IBM WCS* pages

- CMS images and even individual image crops can be used as product images.
- CMS videos can be used as product videos. They will be displayed together with the product images in a gallery.
- CMS content of type `Download` can be offered as additional content that can be downloaded for a product. Any type of binaries are supported, like PDF documents, ZIP archives or office documents.

Such product assets can be edited with *CoreMedia Studio* and will then be delivered by the CMS to enrich, for example, a product detail page.

This section describes the necessary configuration steps for either configuring and deploying *CoreMedia Asset Management* or for removing the contributing modules from the *CoreMedia Blueprint* workspace.

8.7.1 Product Asset Widget

The *Product Asset Widget* can only be used with the e-Commerce blueprint.



To present CMS assets on product detail pages you can replace the default *IBM WCS Full Image Widget* by the *CoreMedia Product Asset Widget* that displays images in an attractive gallery. This makes it particularly easy to present multiple product images and videos for a product.

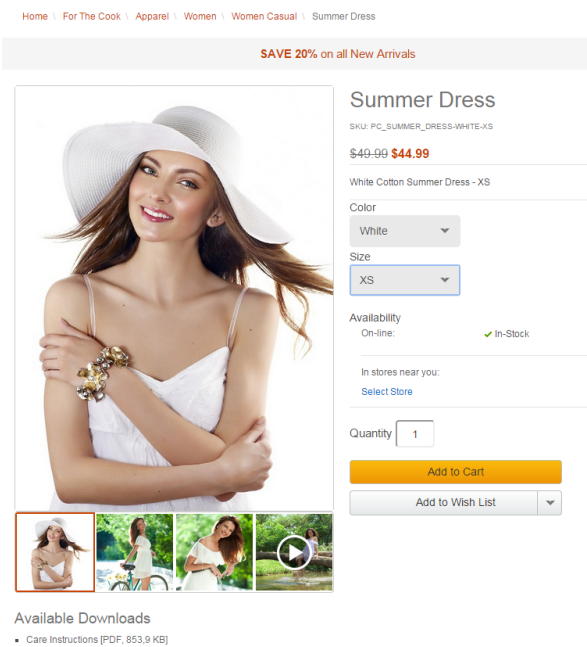


Figure 8.29. Product image gallery delivered by the CMS

The *CoreMedia Product Asset Widget* can also be used to display a list of download links that are associated with the product. The download links are shown together with the product image gallery as *Additional Downloads* or alternatively in a separate slot on the product detail page.

See [Section 3.4.11, “Deploying the CoreMedia Widgets” \[73\]](#) to get the information how to deploy the *CoreMedia Product Asset Widget* and ??? to learn how to use it.

Assign Products to CMS Assets

CoreMedia DXP 8 allows you to manage assets in the CoreMedia system that will be used for products and SKUs in the *IBM WebSphere Commerce Server*.

To achieve this *Picture*, *Video* and *Download* documents can be linked with products. That means one picture, video or download can be (re)used for many products. All images and videos that link to the same product act together as a gallery of images and videos of the same product.

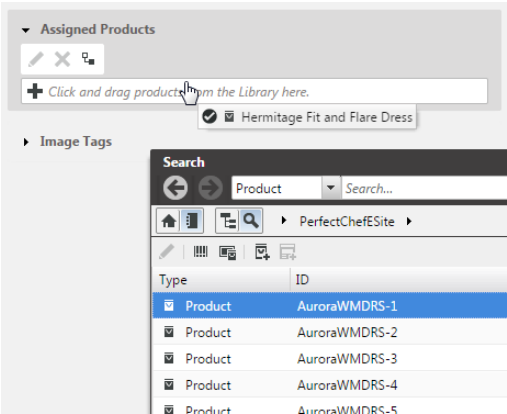


Figure 8.30. Assign a product to a picture

The same applies to downloads. All *Download* documents that link to the same product appear together in an *Available Downloads* list on the product detail page (if the option was used in the *CoreMedia Product Asset Widget*). The order of the images or downloads in the list is determined by the name (in alphabetic order).

You don't have to assign every existing SKU to an asset document, eg. an image, in order to achieve that for each SKU, the same image is delivered. If a SKU is not directly assigned the CMS searches for all asset documents that are assigned to the master product of the SKU or uses the default image for the site (in case of an image).

See Section “Adding Specific Content for Product Detail Pages (PDP)” in *CoreMedia Studio User Manual* to learn how to assign products to images using the *CoreMedia Studio*.

8.7.2 Replaced Product and Category Images

In addition to the *Product Asset Widget* you can replace images directly by replacing the URL in the IBM system with a CoreMedia URL. The linking of product or category images from *IBM WCS* to the *CoreMedia CAE* is done via Image URLs that you can add to the *Display* tab of the product or category definition.

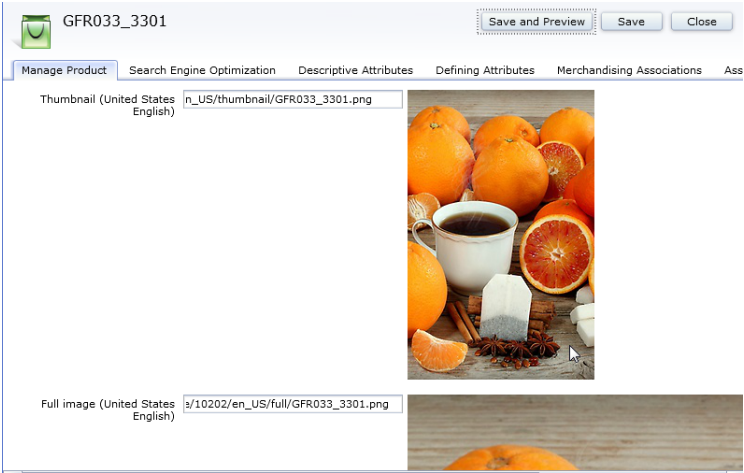


Figure 8.31. Define Product Image URLs in Management Center

Regardless of the usage of the *CoreMedia Product Asset Widget*, once the image URLs of a product are pointing to the CMS all occurrences of these product images (for example, on catalog overview pages) will be delivered by the CMS. If multiple images are assigned to one product, then the first image is taken (in alphabetical order).



The Image URL has the following format:

For a product:

```
http://[cmsHost]/blueprint/servlet/catalogimage/product/  
<StoreId>/<Locale>/<Mapping>/<PartNumber>.jpg
```

For a category:

```
http://[cmsHost]/blueprint/servlet/catalogimage/category/  
[storeId]/[Locale]/<Mapping>/<CategoryId>.jpg
```

Where the path segments have the following meaning:

Segment Name	Example	Description
[cmsHost]	[cmsHost]	The URL prefix of the server that can deliver CMS images. Typically, you will enter here the literal string [cmsHost] so the system can map it to a concrete URL prefix. Since the images are delivered from different servers depending on which side you are (preview or live) the hostname can alter between the systems.

Table 8.14. Path segments in the image URL

Segment Name	Example	Description
		The placeholder [cmsHost] will then be replaced by a URL prefix containing the live host, provided the request comes from the live side. See also the <i>IBM WCS</i> documentation "Configuration properties for content management system integration".
storeId	10202	The ID of the IBM WebSphere Commerce store for which the image is requested. An IBM store is configured for a specific site in the CoreMedia system and the mapping is done via the ID.
Locale	en_US	The locale of the store.
Mapping	thumbnail	The mapping between an image in the IBM product and the named image variant that is taken from the CoreMedia system.
PartNumber/Category-ID	GFR033_3301/PC_ToDrink	The product or SKU part number or category ID.

Delivery of Images

The URL is resolved from the catalog picture handlers. The handlers map the "Named image format" segment to a cropped variant of a picture (see [Section 6.3.14, "Images" \[298\]](#) for details of crops). *CoreMedia Blueprint* comes with the following definition:

```
<bean id="productCatalogPictureHandler"
class="com.coremedia.livecontext.asset.ProductCatalogPictureHandler"

    parent="catalogPictureHandlerBase">
...
<property name="pictureFormats">
  <map>
    <entry value="portrait_ratio20x31/200/310">
      <key>
        <util:constant static-field=
"com.coremedia.livecontext.asset.CatalogPictureHandlerBase.FORMAT_KEY_THUMBAIL"/>
      </key>
    </entry>
    <entry value="portrait_ratio20x31/646/1000">
      <key>
        <util:constant static-field=
"com.coremedia.livecontext.asset.CatalogPictureHandlerBase.FORMAT_KEY_FULL"/>
      </key>
    </entry>
  </map>
</property>
```



```

</bean>

<bean id="categoryCatalogPictureHandler"
class="com.coremedia.livecontext.asset.CategoryCatalogPictureHandler"
    parent="catalogPictureHandlerBase">
    ...
</bean>

```

That is, a URL with a segment thumbnail maps to an image variant portrait_ratio20x31 with the width "200" and the height "310" and a URL with segment full maps to the same image variant portrait_ratio20x31 but with width "646" and height "1000". These are the values required by the IBM Aurora Starter Store.

You can customize the configuration via a Spring configuration as described in [Section "Mapping of Custom Picture Formats" \[451\]](#).

8.7.3 Extract Image Data During Upload

If your pictures files are enriched with the product codes as XMP/IPTC "artwork or object in the picture", the system automatically tries to extract data during the upload. How the data is used depends on the content item to which you upload the image.

- ➔ Upload to a `Picture`: The product codes are extracted and the system tries to add a reference to the product in the e-Commerce repository with this product code.
- ➔ Upload to a `Picture Asset`: The product codes are extracted and are added to the `Picture Asset`.

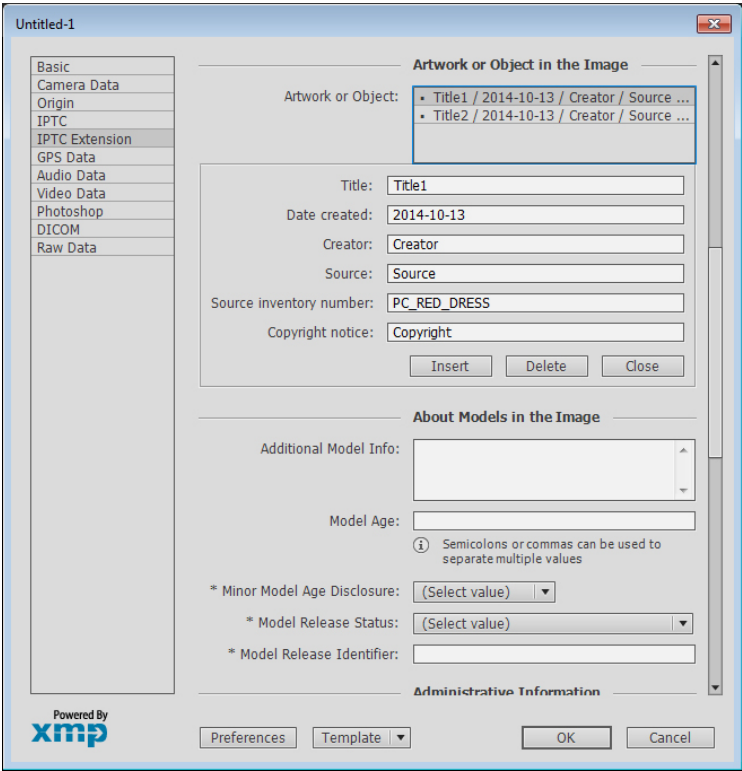


Figure 8.32. Screenshot from Adobe Photoshop for a Picture containing XMP Data

While uploading the pictures via *CoreMedia Studio* into a `Picture` item, the system automatically extracts the product codes and adds references to the assigned products. At this process the product references contained in the original image data will be remembered. You have the option to reset to the original imported data after you have changed the assignments manually.

Upload to a Picture content item

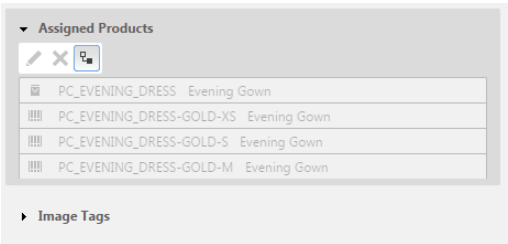


Figure 8.33. Picture linked to XMP Product Reference

After an initial import the status of the *Assigned Products* section is set to "inherited". All associated product references are shown as "read only" and can only be edited if the *Switch off inheritance* button is pressed.

Each re-import of the same image data (with an update of the blob) leads to an update of the associated product references unless the references have been changed manually. In general, the rule applies, that no data will be overwritten that have been changed manually.

8.7.4 Configuring Asset Management

In the following it is described how you can adapt *CoreMedia Asset Management* to your specific needs:

- Define which crops of an image are used in IBM pages.
- Define from which CAEs the IBM system gets images.
- Define content types for your own assets.
- Define publication behavior for renditions of your assets.
- Define where large blobs should be stored.
- Define appropriate rights in the CoreMedia system for your asset content.

Mapping of Custom Picture Formats

This feature can only be used in the e-Commerce blueprint.



You can manage pictures in *CoreMedia DXP 8* that are used in IBM products and SKUs pages. You can use Spring configuration, to map URL path segments to specific crops.

CoreMedia Blueprint comes with a predefined mapping defined in the `catalogPictureHandler` bean. If you want to define your own mapping you can overwrite the default setting as follows:

```
<customize:replace bean="catalogPictureHandler"
id="customizeCatalogPictureHandler"
    property="pictureFormats">
    <description>
        Your custom picture formats for the Catalog Picture Handler
    </description>
    <map>
        <entry key="customFormat1" value="custom_crop1/300/410"/>
        <entry key="customFormat2" value="custom_crop2/700/1200"/>
    </map>
```

```
</customize:replace>
```

The *key* attribute in the *entry* tag is the identifier that is used in the request URL while value is the name of the crop of the image that will be used followed by the size of the image as *"/width/height/* in pixel. The definition of crops is explained in [Section 6.3.14, "Images" \[298\]](#)

Placeholder Resolution for Asset URLs

This feature can only be used in the e-Commerce blueprint.



In the IBM WebSphere Commerce system you can use a placeholder in image URLs which is resolved through a database lookup in the STORECONF table. See the IBM documentation for more details at https://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/refs/rwccmsresolvecontent-tag.htm?lang=en.

For example:

```
http://[cmsHost]:<CAEPort>/blueprint/servlet/catalogimage/product/  
[storeId]/<Locale>/<Mapping>/<PartNumber>.jpg
```

The placeholders in the example above are *[cmsHost]* and *[storeId]*.

To resolve *[cmsHost]* - see the IBM documentation for `ResolveContentURLCmdImpl` for more information. If you want to connect preview and live CAE to one *Management Center* you can define different values for `wc.resolveContentURL.cmsHost` and `wc.resolveContentURL.cmsPreviewHost` in the STORECONF table.

If you use one extended sites catalog for multiple shops you can specify a *[storeId]* placeholder in your image URLs, which are dynamically resolved at runtime.

In a development setup you may share one WCS instance for preview and live delivery.

In order to identify the CAE (preview or live) from which the image should be delivered, the *Blueprint* workspace comes with a predefined Apache configuration. Depending on the shop URL, for example, `shop-helios.blueprint-box.vagrant` versus `shop-preview-helios.blueprint-box.vagrant` the Apache server adds a request header `X-FragmentHost` which contains the value preview or live. For more information about Apache configuration see chapter [Section 3.5.4, "Developing with Apache \(optional for e-Commerce\)" \[91\]](#).

If you want to activate *[cmsHost]* resolution for a shared IBM WCS preview/live environment, perform the following steps:

1. Register and map the `FragmentHostFilter` servlet to workspace/Stores/WebContent/WEB-INF/web.xml of the IBM WCS to extract the X-FragmentHost header information from the request.

```
...
<filter>
  <filter-name>FragmentHostFilter</filter-name>
  <filter-class>com.coremedia.livecontext.servlet.FragmentHostFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>FragmentHostFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

2. Register the `CoreMediaResolveContentURLCmdImpl` in the IBM WCS. This command resolves `[cmsHost]` placeholder in image URLs depending on a preview or live switch for the current request. It resolves `[storeId]` placeholder as well. To register the command perform the following SQL statement:

```
insert into cmdreg (storeent_id, interfacename, classname)
values
(0, 'com.ibm.commerce.content.commands.ResolveContentURLCmd',
'com.coremedia.commerce.content.commands.CoreMediaResolveContentURLCmdImpl');
```

Refer to the IBM documentation for more details about registering custom command implementations in the command registry

To resolve `[storeId]` in Management Center, you have to register and map the `ImageFilter` servlet to workspace/LOBTools/WebContent/WEB-INF/web.xml of the IBM WCS.

```
...
<filter>
  <filter-name>ImageFilter</filter-name>
  <filter-class>com.coremedia.livecontext.servlet.ImageFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ImageFilter</filter-name>
  <url-pattern>/LoadImage</url-pattern>
</filter-mapping>
...
```

Content Types

CoreMedia Advanced Asset Management stores its data in the content repository in content items. *CoreMedia DXP 8* contains the abstract root content type `AMAsset` (see Chapter 4, *Developing a Content Type Model* in *CoreMedia Content Server Manual* for a description of content types) as a starting point for assets. `AMAsset` defines a property `original` to store the raw editable form of the asset and another property `thumbnail` to store a thumbnail view. The thumbnail property can be

Abstract content type
`AMAsset`

used for a uniform preview of assets. If there is no sensible thumbnail for an asset, it can be left empty.

Concrete content types for specific assets, such as pictures or documents, need to extend the abstract content type `AMAsset`. Most probably, you will add more properties for different renditions of the asset. Names of rendition properties must be alphanumeric strings. By default, `AMPictureAsset` and `AMDocumentAsset` are provided as a non-abstract asset type, defining rendition properties for web delivery and for printing.

*Concrete content types
AMPictureAsset and
AMDocumentAsset*

You can modify existing asset types or define additional asset types in the file `asset-management-plugin-doctypes.xml` in the `am-server` module. For each asset type, you need an appropriate form in *CoreMedia Studio*. *CoreMedia Blueprint* already defines suitable *Studio* forms for the `AMPictureAsset` and `AMDocumentAsset`. Change this form when you adapt the `AMPictureAsset` or `AMDocumentType` content type and add further forms for your own asset types.

*Defining your own as-
set types*

When you add further rendition properties that hold very large blobs, modify the blob store configuration as described in Section 3.4, “Configuring Blob Storage” in *CoreMedia Content Server Manual*. Small renditions up to a few megabytes can be stored in the *Content Server* database and do not need additional configuration.

*Store large blobs in the
file system*

To prevent large blobs like the original rendition from being published, you can exclude them from publication process. For more information read [Section “Configure Rendition Publication” \[454\]](#).

Configure Rendition Publication

Certain renditions can be excluded from publication. To do so the `am-server-component` comes with an `AssetPublishInterceptor` which reads the `metadata` property of assets to determine if a given rendition should be published or not.

The `AssetPublishInterceptor` bean is added to the *Content Server* and to the corresponding command-line tools. The following properties control the behavior of the interceptor:

assetMetadataProperty

The `Struct` property which contains the information whether to publish a rendition or not at path `renditions.<rendition-name>.show`. If the Boolean property `show` is `true` the rendition blob will be published. Otherwise the blob will not be available on the master server.

interceptingSubtypes

Boolean flag to control whether also subtypes of `type` should be intercepted or not.

removeDefault

The default value to control whether a rendition blob should be removed from publication or not. If unset the default is to remove blobs if nothing else is specified in either the metadata struct or in the removal overrides.

removeOverride

Overrides any setting or default for a given rendition. It contains a map from rendition name to removal flag. Thus if you want the rendition `thumbnail` to be published in any case add an entry with key `thumbnail` and value `false`.

type

The document type the interceptor applies to. For subtype processing set the flag `interceptingSubtypes` accordingly.

Example 8.9, “Rendition Publication Configuration” [455] shows a possible configuration of the `AssetPublishInterceptor`.

```
<beans ...>
  <util:map id="removeOverride"
    key-type="java.lang.String"
    value-type="java.lang.Boolean">
    <entry key="thumbnail" value="false"/>
  </util:map>

  <bean id="assetPublishInterceptor"
    class=
      "com.coremedia.blueprint.assets.server.AssetPublishInterceptor">

    <property name="type" value="AMAsset"/>
    <property name="interceptingSubtypes" value="true"/>
    <property name="assetMetadataProperty" value="metadata"/>
    <property name="removeDefault" value="true"/>
    <property name="removeOverride" ref="removeOverride"/>
  </bean>
</beans>
```

Example 8.9. Rendition Publication Configuration

Blob Storage

Blobs of renditions can be stored in the database or in the file system. In general, content in the CoreMedia CMS is stored in a database, but for large blobs, the file system might be better suited for storage, because databases are not always optimized for this use case.

Blueprint is configured in such a way, that WAR files for deployment, generated in the `packages` directory, are configured to store the blobs of the default asset type in the file system. However, when you start the *Content Management Server* using

Blueprint default storage behavior

`mvn tomcat7:run-war` in module `content-management-server-webapp` of the development installation, all blobs are stored in the database.

Blob storage is controlled by an XML file (default is `blobstore.xml`), which defines the storage configuration (see Section 3.4, “Configuring Blob Storage” in *CoreMedia Content Server Manual* for details). With the property `am.blobstore.rootdir`, you define the root directory for file system storage. The value of the property is used in the XML file.

Configuration of blob storage

In *Blueprint* you find the storage configuration in the file `src/main/resources/framework/spring/blobstore/am/blobstore.xml` of the `am-server` module. The property `am.blobstore.rootdir` can be set in the file `application.properties` of the *Content Management Server* web application or as a system property. File and property must both be present, so that file system storage works. In the packaged WAR files from the *Blueprint* packages directory, the property `am.blobstore.rootdir` is set to `DATA_ROOT/cm7-cms-tomcat/blobstore/assets`, where `DATA_ROOT` is the base directory for storing long-lived application files. `DATA_ROOT` defaults to `/var/lib/coremedia`. These properties can be adapted in the file `blueprint/packages/pom.xml` as needed.

Blueprint default configuration

If you want to store your own renditions in the file system, update the blob store configuration accordingly. If you want to store them in a separate database, you have to define an appropriate media store, as described in Section 3.4, “Configuring Blob Storage” in *CoreMedia Content Server Manual*. If you want to store assets in the content repository database, remove the configuration file and reset `am.blobstore.rootdir`. If you want to store them in a separate database, define an appropriate media store.

Keep in mind, that storing a blob in the file system might double the required space, when you use the rendition in another content item, for example, in a `Picture`.

This is because, when you store a blob in the database and the same blob is used in different content items, then all the content items link to this blob. On the other hand, when you have stored a blob in the file system and this blob is used in another content item that does not define file system storage, then a copy of the blob will be created in the database.



Rights

Assets in the form of `AMAsset` documents are placed in the `/Assets` folder by default. Define rights rules for the content repository in such a way that only authorized users can create and change assets and that assets can only be placed in the folder `/Assets`. Note that access rights for the root content type `Document_` automatically imply rights on assets.

Studio

The asset management extension of *CoreMedia Studio* is defined in the modules `am-studio` and `am-studio-component`.

In `am-studio` you can find the form definition for picture forms in the file `AMPictureAssetForm.xml`. Update this file if you change the set of renditions. Create additional form when you add further asset types. Localizations of asset types and rendition names can be added to the resource bundle `AMDocumentTypes`.

The module `am-studio-component` contains configuration information for the Studio REST backend. In the file `component-am-studio.xml` you can find the configuration of two write interceptors which update the asset metadata as renditions are uploaded using Studio.

Asset Download Portal

CoreMedia Advanced Asset Management comes with an asset download portal. You can configure the behavior of the portal in the Asset Management Configuration content item in *Studio* as shown in [Figure 8.34, “Configuration of the download portal”](#) [457].

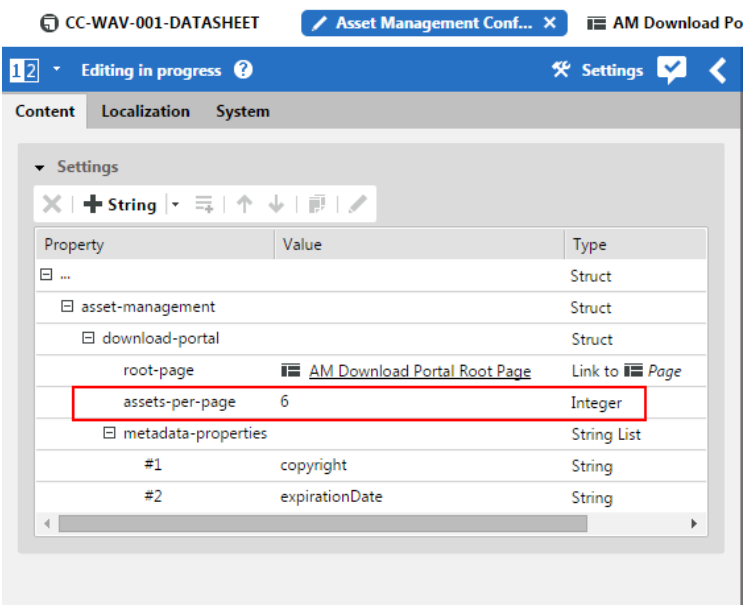


Figure 8.34. Configuration of the download portal

The properties in the `download-portal` struct have the following meaning:

- root-page

A `Page` content item which defines the context of the download portal. The root page contains the *AM Download Portal Placeholder* in the *Main* placement.
- assets-per-page

The number of assets that are shown in own page.
- metadata-properties

The properties from the asset's metadata that are shown in the detail view of an asset.

The hierarchy of the assets in the download portal is determined by the Asset Download Portal taxonomy. That is, an asset content item is shown on the download portal, when it contains an asset category tag and a downloadable property.

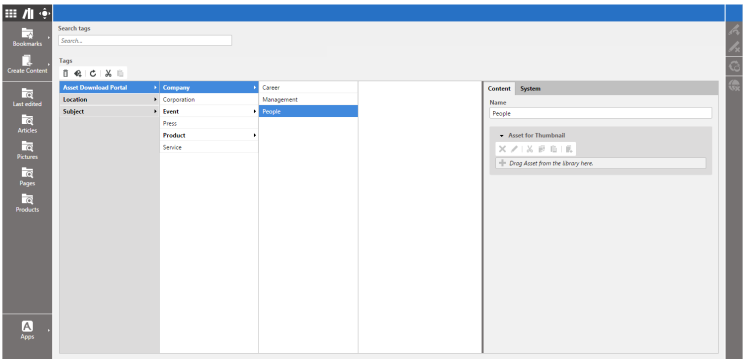


Figure 8.35. Taxonomy for assets

8.7.5 Using the Adobe Drive Connector

Assets can be presented in a virtual file system using the Adobe Drive integration. Thus, you can access assets stored in the CoreMedia system from your Adobe applications. To make use of this feature, a server web application must be deployed and the Adobe Drive installation must be configured.

Mapping CoreMedia content and folder on the file system

CoreMedia Asset Management stores assets as content in the content repository. All assets are stored below a base folder, typically named `/Assets`.

For the file system mapping, CoreMedia folders are represented as directories with the same name as the folder. Asset contents are represented as directories with a special name prefix to distinguish them from folders. In the default configuration each picture asset is prefixed with `Asset Picture` and each document asset with `Asset Document`. For example, an asset `Foo` would be represented by the directory `Asset Picture Foo`. Each asset directory contains one file for each rendition. In the above example the directory might contain the files `Foo_original.psd`,

File system mapping

`Foo_thumbnail.jpg` and `Foo_web.jpg` holding the original asset, the thumbnail and the web rendition, respectively.

It is not supported to change the type of an asset by renaming the asset directory using a different prefix. It is not supported that two different users check out renditions of one asset at the same time. It is not supported to add further rendition files that do not indicate one of the configured rendition properties in their name.



Adobe Drive Web Application

The extension module `am-adobe-drive-server-webapp` contains the web application that serves as the backend for the Adobe Drive integration. For a compact deployment you add this web application to the Studio Tomcat, modifying the file `blueprint/packages/services/studio-tomcat/pom.xml` to include a dependency and a configuration of the context name. Alternatively, a new application container can be created that is dedicated to the Adobe Drive connector web application.

The file `application.properties` of the module `am-adobe-drive-server-webapp` contains configuration information about the server location and the content type model. In any case, you need to configure at least the reference to the *Content Management Server* in the property `repository.url` and the password of the `webserver` user in the property `repository.password`.

The property `cm.assets.assetBaseFolderPath` can be used to specify the root folder of the content repository for storing assets. Typically the default `/Assets` is appropriate.

Further properties in the file `application.properties` specify various aspects of the content type model like the names of the properties storing thumbnails, previews or metadata. See the configuration file for details about these properties.

In the file `src/main/webapp/WEB-INF/application.xml` of the `am-adobe-drive-server-webapp` module you can configure the naming scheme for individual asset types. For each asset type you can configure the name of the content type storing such assets, the name pattern for directories representing entire assets and the name pattern for file representing individual renditions.

Naming scheme for asset mapping

The following XML fragment shows the default configuration of the picture asset type:

```
<bean id="pictureAssetMapping"
class="com.coremedia.cms.assets.drive.config.AssetTypeMapping">

  <property name="contentType" value="AMPictureAsset"/>
  <property name="folderPattern" value="Asset Picture {0}"/>
  <property name="filePattern" value="{0}_{1}.{2}"/>
</bean>
```

In the name patterns, the placeholder {0} refers to the asset document name. {1} represents the name of the property storing the rendition. {2} is the user-chosen file extension. When parsing file names, the parameters {1} and {2} match alphanumeric strings, only.

Placeholders for naming pattern

Directory patterns must be unambiguous, that is, no conceivable directory name may match two different directory patterns. The file pattern must contain all three placeholders, ending in the file extension placeholder {2}.

Modify the given pattern and add further patterns as needed. Update the mapping when you update the content type schema.

Adobe Drive Connector

Before you can install Adobe Drive, you need to have an Adobe client installed, such as Photoshop or Bridge. When you have installed Adobe Drive, add the CoreMedia Drive Connector as follows:

1. Download the `com.coremedia.adobe.adobe-drive-client.bundle.jar` file from the CoreMedia Adobe Drive Connector REST back-end. Simply enter the connection URL to the back-end into your browser. Click the *Download Client Bundle* link on the Welcome page.
2. Stop Adobe Bridge or any other Adobe client application, Adobe Drive and the background process `AD4ServiceManager`.

When the configuration of the CoreMedia Adobe Drive Connector REST back-end was changed, or when the content IDs in the CoreMedia repository have changed (for example, due to re-import in a development or QA system), delete the following cache folders:

➔ Windows:

```
C:\Users\USERNAME\AppData\Roaming\Adobe\AD4ServiceManager\database
```

```
C:\Users\USERNAME\AppData\Roaming\Adobe\AD4ServiceManager\diskcache
```

➔ Mac:

```
/Users/USERNAME/Library/Application Support/Adobe/AD4ServiceManager/database
```

```
/Users/USERNAME/Library/Application Support/Adobe/AD4ServiceManager/diskcache
```

3. Add the file `com.coremedia.adobe.adobe-drive-client.bundle.jar` to the plugin directory of Adobe Drive. The directory can be found at the following locations:

→ Windows:

```
C:\Program Files\Common Files\Adobe\AD4ServiceManager\plugins
```

→ Mac:

```
/Library/Application Support/Adobe/AD4ServiceManager/plugins/
```

4. Restart Adobe Drive.

Now, you can add a new connection using the *CoreMedia* connector. You are asked to configure a remote URL, which is the root URL of the Adobe Drive web application. Also enter your user name and password for the CoreMedia system.

It is highly recommended to use a secure connection between the Adobe Drive client and the Adobe Drive web application. In order to establish an SSL connection, the client needs to verify the full certificate chain of the server certificate. If you encounter any issues with this verification, please refer to the Adobe Drive Admin Guide (issued by Adobe).

When you do not use the official certificates, in a CI, for instance, the untrusted certificate chains have to be added to the Java (v1.6) truststore (default password: `changeit`) of the Adobe Drive client. The easiest way to retrieve the certificates is using Firefox and the connection information dialog in the address bar **View Certificate|Details|Export**. Add the certificates as follows (path for Mac OS, replace the names in angle brackets with your own values):

```
sudo keytool -import -file <mycertificate.cer> -alias <myalias> -keystore
/System/Library/Java/Support/CoreDeploy.bundle/Contents/Home/lib/security/cacerts
```



Example 8.10. Adding certificates to truststore

9. Appendix

This chapter contains detailed information about some of *CoreMedia Blueprint's* features:

- [Section 9.1, “Port Reference” \[463\]](#) shows the deployment of the CoreMedia components in the boxes with their default ports.
- [Section 9.2, “Typical LiveContext Deployment” \[467\]](#) shows a typical LiveContext deployment with the required ports.
- [Section 9.3, “Linux / Unix Installation Layout” \[468\]](#) shows the default file installation layout of CoreMedia components.
- [Section 9.5, “Maven Profile Reference” \[475\]](#) shows the predefined profiles in the *CoreMedia Blueprint* workspace.
- [Section 9.6, “Content Type Model” \[476\]](#) shows UML diagrams of *CoreMedia Blueprint* content types.
- [Section 9.7, “Link Format” \[478\]](#) lists the controller, link schemes and link post-processors of *CoreMedia Blueprint*.
- [Appendix - Predefined Users \[484\]](#) shows the predefined users that are available in the system for log in to *Studio*.
- [Section 9.9, “Database Users” \[487\]](#) shows the database users that are needed by the CoreMedia server components.
- [Section 9.10, “Cookies” \[488\]](#) lists all cookies delivered by *CoreMedia DXP 8*.

9.1 Port Reference

For each application or component there is a two digit port prefix defined and for each protocol or service there is a three digit port suffix defined.

The next figure shows the deployment of the *CoreMedia DXP 8* components into the boxes and the communication between the components. If you cannot read the text of the figure in the online documentation, right-click the image and select Open image in new tab from the context menu. You can also download the original YEd file from the documentation overview page below *Other Documentation* named *CoreMedia DXP 8 Deployment Diagram*.

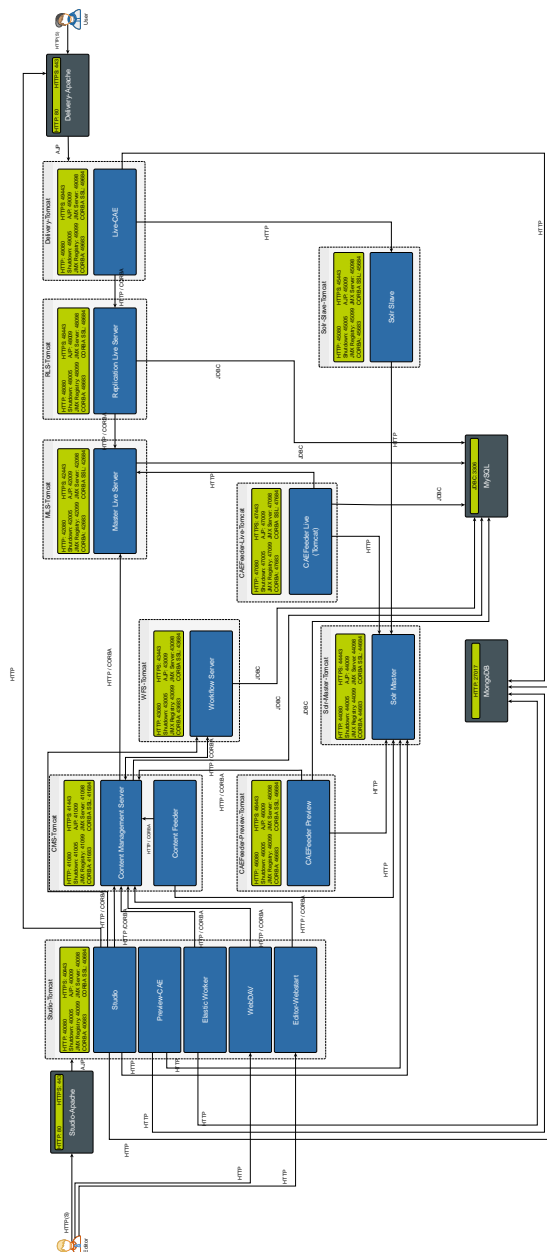


Figure 9.1. Deployment and communication overview

Component	Port Prefix
Studio	40
Content Management Server	41
Master Live Server	42
Workflow Server	43
Solr Master	44
Solr Slave	45
CAE Feeder Preview	46
CAE Feeder Live	47
Replication Live Server	48
Delivery (Live CAE)	49

Table 9.1. Component Port Prefix

Protocol / Service	Port Suffix
HTTP	080
HTTPS	443
Tomcat Shutdown	005
Tomcat AJP	009
Tomcat JMX Registry Port (RMI)	099
Tomcat JMX Server Port (RMI)	098
Tomcat Debug Port	777
CORBA	683
CORBA SSL	684

Table 9.2. Protocol / Service Port Suffix

Since the preview is always deployed together with *Studio*, there is no extra port prefix reserved. When started with `tomcat7:run` from within the workspace, the preview Tomcat process has its own HTTP port. If you want to access the preview, you have to use the port 40081.



Service	Port
MySQL	3306

Table 9.3. Third-Party Services

Service	Port
Mongo DB	27017



Be advised, that this port schema's port range has its drawback of overlapping with the default ephemeral port range on most operation systems. You can however, decrease the ephemeral port range with the cost of not optimizing your TCP/IP stack. In a setup with just one CAE running per host, decreasing the range will likely not affect your systems performance, but if you are running many CAEs on one host, you should rather adapt the port filtering to use a smaller range of ports below the default ephemeral port range.

Changing the filtering can be done in root `pom.xml` using the port schema properties or in the `packages/pom.xml` using the explicit properties. There should be no hard coded ports beside the *Chef* test specs, which are only active in the *Vagrant* virtualized development mode. If there are hard coded ports in property files it is most likely, that there is also a configurable override in the Tomcat plugin configuration of the `pom.xml` or in the override properties in the packages hierarchy.

To decrease the ephemeral port range for CentOS, you just need to execute the following commands as root on your system:

```
echo "50000 65535" > /proc/sys/net/ipv4/ip_local_port_range
echo -e "\n# Increase system IP port
limits\nnet.ipv4.ip_local_port_range = 50000 65535" >>
/etc/sysctl.conf
```

9.2 Typical LiveContext Deployment

Figure 9.2, “Typical deployment and ports of a LiveContext system” [467] shows a typical deployment of CoreMedia DXP in combination with the IBM WebSphere Commerce server in a commerce-led scenario (see [Section 5.1, “Commerce-led Integration Scenario”](#) [176]).

If you cannot read the text of the figure in the online documentation, right-click the image and select Open image in new tab from the context menu. You can also download the original YEd file from the documentation overview page below *Other Documentation* named *CoreMedia DXP 8 Deployment Diagram*.

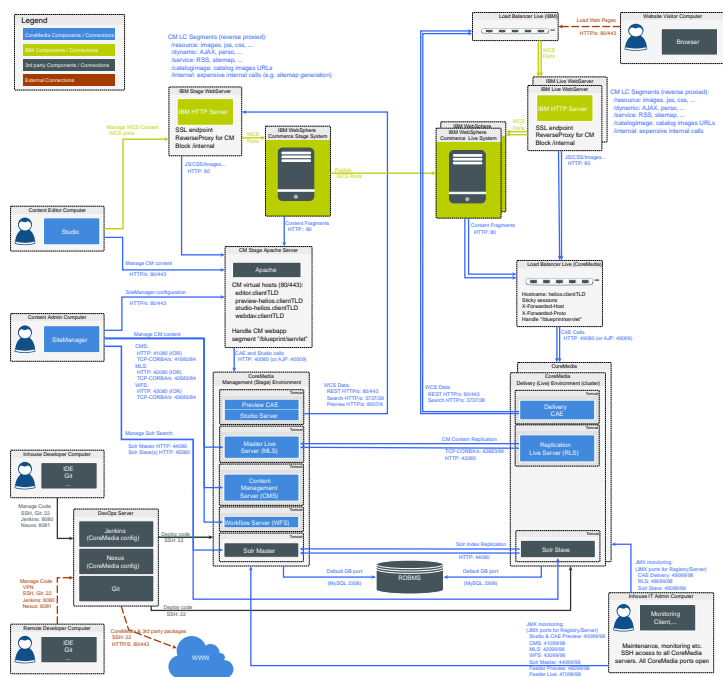


Figure 9.2. Typical deployment and ports of a LiveContext system

9.3 Linux / Unix Installation Layout

Since RPM and other native packaging formats don't just archive files but merely archive them with the target destination path, you have to think about the installation layout at build time. You will find the layout directives in the `pom.xml` files below the `packages` module hierarchy.

CoreMedia DXP 8 strives to standardize the file layout on a target machine. The table below lists the common paths where configuration, log files or scripts can be found.

Table 9.4. Default Package Layout

Default Path	Usage
/etc/coremedia	Each package will install and require a configuration property file below this folder. To reconfigure a service use the service initialization script and its reconfigure <i>option</i> , for example <code>sudo service cm7-cms-tomcat reconfigure</code> . To reconfigure a tool a shell script will be provided in the tools installation directory. Although not recommended, to change the default, modify the Maven property <code>CONFIGURE_ROOT</code> in the <code>packages/pom.xml</code> .
/etc/coremedia/APPLICA TION_NAME.properties	The configuration file to configure or reconfigure a package.
/opt/coremedia	The installation directory for all CoreMedia packages. To change the default, modify the Maven property <code>INSTALL_ROOT</code> in the <code>packages/pom.xml</code>
/var/log/coremedia	The logging directory for all CoreMedia packages. To change the default, modify the Maven property <code>LOG_ROOT</code> in the <code>packages/pom.xml</code>
/var/run/coremedia	The directory where all Tomcat services will create their PID files. To change the default, modify the Maven property <code>PID_ROOT</code> in the <code>packages/pom.xml</code>
/var/cache/coremedia	The common directory where Tomcat services will create their temporary folders. To change the default, modify the Maven property <code>TMP_ROOT</code> in the <code>packages/pom.xml</code>
/var/lib/coremedia	The common directory where Tomcat services store persistent data if the data is not put into a database. To change the default, modify the Maven property <code>DATA_ROOT</code> in the <code>packages/pom.xml</code>

Default Path	Usage
/opt/coremedia/cm7-tomcat-installation	The directory of the common Tomcat installation
/etc/init.d/APPLICATION_NAME	The Tomcat service file. By default, the services file defines the following methods (start, stop, status, restart, reload, reconfigure).
/var/log/APPLICATION_NAME.out	The Tomcat standard out file.
/etc/coremedia/APPLICATION_NAME.conf	The Tomcat service configuration. In this file you can modify CATALINA_HOME, CATALINA_PID, CATALINA_TMPDIR and CATALINA_OUT

9.4 IBM WebSphere Commerce REST Services used by CoreMedia

CoreMedia Digital Experience Platform 8 uses REST services of the IBM WebSphere Commerce Server Management Center to access content. Here you find a list of URLs used by Studio and CAE.

REST Services used by CoreMedia Studio

The following REST services are only used in combination with IBM WCS (FEP7+)

FEP7+

- ➔ `http://<search_server>/search/resources/store/<storeId>/categoryview/@top`
- ➔ `http://<search_server>/search/resources/store/<storeId>/categoryview/%20?categoryIdentifier=<categoryIdentifier>`

This search-based REST call allows slash character in the category identifier.

- ➔ `http://<search_server>/search/resources/store/<storeId>/categoryview/byId/<uniqueId>`
- ➔ `http://<search_server>/search/resources/store/<storeId>/categoryview/byParentCategory/<uniqueId>`
- ➔ `http://<search_server>/search/resources/store/<storeId>/productview/byCategory/<categoryId>`
- ➔ `http://<search_server>/search/resources/store/<storeId>/productview/bySearchTerm/<term>`

The following REST services are only used in combination with IBM WCS (FEP6)

FEP6

- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/@top`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/<categoryIdentifier>`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/byId/<uniqueId>`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/byParentCategory/<uniqueId>`

- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/productview/byCategory/<categoryId>`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/productview/bySearchTerm/<term>`

The following REST services are used no matter what feature pack is used.

- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/spot/<spotId>`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/spot`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/segment/<uniqueId>`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/segment`
- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/workspaces/byall/Active`

Only used if customer uses IBM WCS workspaces

- ➔ `http://<wc_server>/wcs/resources/coremedia/cacheinvalidation/latestTimestamp`

CoreMedia specific custom REST service

- ➔ `http://<wc_server>/wcs/resources/coremedia/cacheinvalidation/<timestamp>`

CoreMedia specific custom REST service

- ➔ `http://<wc_server>/wcs/resources/coremedia/languagemap`

Used to map langId to numeric value

- ➔ `http://<wc_server>/wcs/resources/coremedia/storeinfo`

Used to get the storeId and the catalog information from all available stores in IBM WCS

REST Services used by the CAE

The following REST services are only used in combination with *IBM WCS (FEP8+)*



- ➔ `http://<wc_server>/wcs/resources/store/<storeId>/price?q=byPartNumbers&partNumber=<partNumber>`

FEP7+

The following REST services are only used in combination with *IBM WCS (FEP7+)*

→ `http://<search_server>/search/resources/store/<storeId>/categoryview/%20?categoryIdentifier=<categoryIdentifier>`

This search-based REST call allows slash character in the category identifier.

→ `http://<search_server>/search/resources/store/<storeId>/categoryview/<SeoSegment>`

→ `http://<search_server>/search/resources/store/<storeId>/categoryview/byId/<uniqueId>`

→ `http://<search_server>/search/resources/store/<storeId>/productview/%20?partNumber=<productIdentifier>`

This search-based REST call allows slash character in the product identifier.

→ `http://<search_server>/search/resources/store/<storeId>/productview/byId/<uniqueId>`

→ `http://<search_server>/search/resources/store/<storeId>/productview/bySearchTerm/<term>`

FEP7

The following REST services are only used in combination with *IBM WCS (FEP7)*

→ `http://<wc_server>/wcs/resources/store/<storeId>/price`

FEP6

The following REST services are only used in combination with *IBM WCS (FEP6)*

→ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/<categoryIdentifier>`

→ `http://<wc_server>/wcs/resources/store/<storeId>/categoryview/byId/<uniqueId>`

→ `http://<wc_server>/wcs/resources/store/<storeId>/productview/<productIdentifier>`

→ `http://<wc_server>/wcs/resources/store/<storeId>/productview/byId/<uniqueId>`

→ `http://<wc_server>/wcs/resources/store/<storeId>/productview/bySeo/<languageIdentifier>/<siteIdentifier>/<productIdIdentifier>`

The following REST services are used no matter what feature pack is used.

→ `http://<wc_server>/wcs/resources/store/<storeId>/espot/<espotIdentifier>`

→ `https://<wc_server>/wcs/resources/store/<storeId>/loginidentity`

→ `https://<wc_server>/wcs/resources/store/<storeId>/previewToken`

→ `http://<wc_server>/wcs/resources/store/<storeId>/inventoryavailability/<productIdsAsCSV>`

→ `http://<wc_server>:<searchport>/search/resources/store/<storeId>/productview/%20?partNumber=<productIdIdentifier>`

This search-based REST call allows slash character in the product identifier.

→ `http://<wc_server>/wcs/resources/store/<storeId>/usercontext/@self/contextdata`

Used by *Elastic Social*

→ `https://<wc_server>/wcs/resources/store/<storeId>/person/@self`

Used by *Elastic Social*

→ `https://<wc_server>/wcs/resources/store/<storeId>/segment`

Used by *Adaptive Personalization*

→ `http://<wc_server>:<searchport>/search/resources/store/<storeId>/sitecontent/keywordSuggestionsByTerm/dres`

→ `http://<wc_server>/wcs/resources/store/<storeId>/cart/@self`

→ `http://<wc_server>/wcs/resources/coremedia/cacheinvalidation/latestTimestamp`

CoreMedia specific custom REST service

→ `http://<wc_server>/wcs/resources/coremedia/cacheinvalidation/<timestamp>`

CoreMedia specific custom REST service

→ `http://<wc_server>/wcs/resources/coremedia/languagemap`

Used to map langId to numeric value

→ `http://<wc_server>/wcs/resources/coremedia/storeinfo`

Used to get the storeId and the catalog information from all available stores in IBM WCS

9.5 Maven Profile Reference

There are some profiles predefined in the workspace. Some are relevant for building and some are relevant for starting components. To activate them simply add them to your profiles list, such as `mvn clean install -Pvagrant`.

Table 9.5. Maven profiles

Profile name	Description
integration-test	This profile activates integration tests. If activated, tests will run that require a running system.
internal-licenses	This profile is used to package license files together with the server web applications. It defines a dependency to a war artifact, which should contain a <code>WEB-INF/properties/corem/license.zip</code> . For more information see section “Configuring the licenses” [47]
performance	This profile is located in the <code>modules/cae/cae-performance-test</code> module and activates <i>JMeter</i> performance tests, if configured.
postconfigure	This profile is the counterpart of the <code>preconfigure</code> Profile, it tells <i>Maven</i> to insert tokens into the configuration files of the packages for post-configuration on the target machine at installation time, see Section 4.3.9, “Configure Filtering in the Workspace” [171] for a detailed description of this process.
preconfigure	This profile tells <i>Maven</i> to load the <code>packages/src/main/filters/preconfigure.properties</code> , to preconfigure the RPM or Zip files for deployment. Artifacts build this way can only be deployed on the system the <code>preconfigure.properties</code> defines for.
vagrant	To develop against the databases and servers installed on the virtualized environment provided with <i>Chef</i> and <i>Vagrant</i> activate this profile. It overrides the default values for the host properties.
s3-upload	Creates ZIP archives for the RPM repository and the content and uploads them to Amazon S3. This profile is only relevant for the boxes module and only if you want to deploy a test system to the cloud.
repository-upload	Collects and uploads the RPMs to a Maven repository. Use this profile together with a <i>Maven</i> repository server that is capable of serving <i>Yum</i> repositories. This profile is only relevant for the boxes module and can be activated during the <code>perform</code> phase of the <i>Maven</i> release process.

9.6 Content Type Model

This section shows the content types of *CoreMedia Blueprint* as UML diagrams. Since the content type model exists of more than forty items it is split into the following diagrams:

- [Figure 9.3, “CoreMedia Blueprint Content Type Model - CMLocalized”](#) [476] shows the content types inheriting from CMLocalized.
- [Figure 9.4, “CoreMedia Blueprint Content Type Model - CMNavigation”](#) [477] shows the content types inheriting from CMNavigation.
- [Figure 9.5, “CoreMedia Blueprint Content Type Model - CMHasContexts”](#) [477] shows the content types inheriting from CMHasContexts.
- [Figure 9.6, “CoreMedia Blueprint Content Type Model - CMMedia”](#) [477] shows the content types inheriting from CMMedia.
- [Figure 9.7, “CoreMedia Blueprint Content Type Model - CMCollection”](#) [477] shows the content types inheriting from CMCollection.

The following diagrams contain most of the content types. The colors have the following meaning:

- Blue items are part of the basis Blueprint content items
- Yellow items are part of the WebSphere Commerce integration
- Green Items are part of the Adaptive Personalization Integration
- Red items are part of the Elastic Social Integration
- Gray items are part of the Analytics Integration

You can download the complete diagram as a `graphml` file from the online documentation page below *Other Documentation* named *CoreMedia DXP 8 Content Type Diagram*:

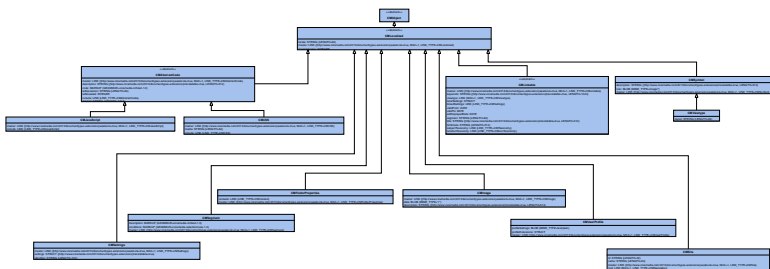


Figure 9.3. CoreMedia
Blueprint Content Type
Model - CMLocalized

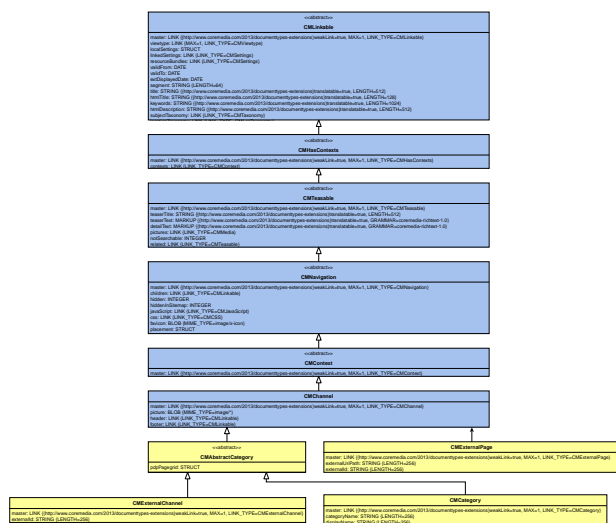


Figure 9.4. CoreMedia
Blueprint Content Type
Model - CMNavigation

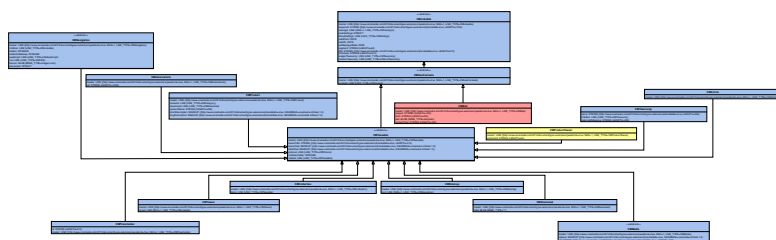


Figure 9.5. CoreMedia Blueprint Content Type Model - CMHasContexts

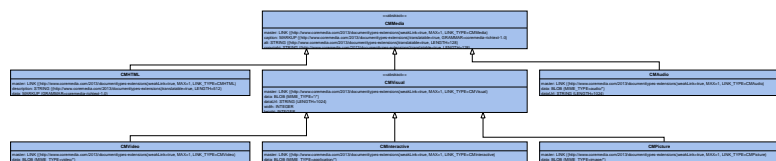


Figure 9.6. CoreMedia
Blueprint Content Type
Model - CMMedia

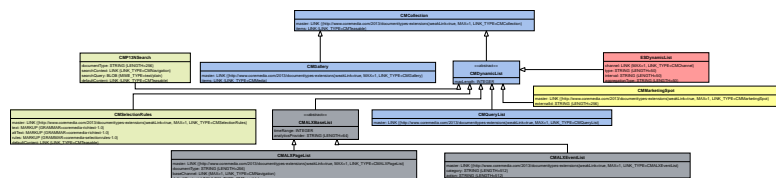


Figure 9.7. CoreMedia
Blueprint Content Type
Model - CMCollection

9.7 Link Format

The following table summarizes most of the corresponding link schemes and controllers of *CoreMedia Blueprint* as defined in `framework/spring/blueprint-handlers.xml`. See the Javadoc of the respective classes for further details.

CapBlobHandler	
Description	Controller and link scheme for Blobs like Images in CSS or Images that do not have any scaling information.
Class	<code>com.coremedia.blueprint.cae.handlers.CapBlobHandler</code>
Example	<code>/blob/1784/4fb7741a1080d02953ac7d79c76c955c/media-data.ico</code> for a CSS background image

Table 9.6. *CapBlobHandler*

CodeResourceHandler	
Description	Controller and link scheme for CSS and JavaScript stored in the CMS.
Class	<code>com.coremedia.blueprint.cae.handlers.CodeResourceHandler</code>
Example	<code>/code/1214/5/responsive-css.css</code> for a CSS

Table 9.7. *CodeHandler*

ExternalLinkHandler	
Description	A Link scheme for external links stored in the CMS.
Class	<code>com.coremedia.blueprint.cae.handlers.ExternalLinkHandler</code>
Example	<code>http://www.coremedia.com</code>

Table 9.8. *ExternalLinkHandler*

PageActionHandler	
Description	Controller and link scheme for <code>CMAAction</code> beans which are for example used to perform a search.
Class	<code>com.coremedia.blueprint.cae.handlers.PageActionHandler</code>

Table 9.9. *PageActionHandler*

Example	/action/corporate/4420/action/search for performing a search
---------	--

PageHandler	
Description	Controller and link scheme for pages.
Class	com.coremedia.blueprint.cae.handlers.PageHandler
Example	/corporate/for-professionals/services for an service page.

Table 9.10. PageHandler

PreviewHandler	
Description	Controller and link scheme previewing content in CoreMedia Studio.
Class	com.coremedia.blueprint.cae.handlers.PreviewHandler
Example	/preview?id=coremedia:///cap/content/3048%26view=fragmentPreview for preview content as a fragment

Table 9.11. PreviewHandler

StaticUrlHandler	
Description	Controller and link scheme for generating static URLs based on Strings
Class	com.coremedia.blueprint.cae.handlers.StaticUrlHandler
Example	/elastic/social/ratings for a ES Post Controller

Table 9.12. StaticUrlHandler

TransformedBlobHandler	
Description	Controller and link scheme for transformed blobs
Class	com.coremedia.blueprint.cae.handlers.TransformBlobHandler

Table 9.13. TransformedBlobHandler

Example	<code>/image/3126/landscape_ra tio4x3/349/261/971b670685dff69cfd28e55177d886db/Pi/mom basa-image-image.jpg</code>
----------------	---

Link Post Processors

While link schemes are responsible for the path and possibly the parameters of a resource's URL, they are not aware of deployment aspects like domains, hosts, ports, servlet contexts, rewrite rules and the like. The Blueprint uses Link Post Processors to format links according to the particular environment.

The following link post processors are applied in `com.coremedia.blueprint.base.links.impl.BaseUriPrepender` and `com.coremedia.blueprint.base.links.impl.LinkAbsolutizer`.

→ `prependBaseUri`

Prepends the "base URI" (web application and mapped servlet, for example `/blueprint/servlet`) to ALL (annotation based) links. This is required when the CAE web application is served directly by a web container with no prior URL rewriting.

→ `makeAbsoluteUri`

Adds a prefix that makes the URI absolute. There are several cases in which URLs must be made absolute:

- a cross-site link: a URI pointing to a resource in a site that is served on a different domain
- externalized URIs: a URI should be send by mail or become part of an RSS feed

The prefixes for absolute URLs are specific for each site, therefore they are maintained in each site's settings in a struct named `absoluteUrlPrefixes`. The prefixes are different for the live and the preview CAE and must be maintained independently. A typical `absoluteUrlPrefixes` struct looks like this:

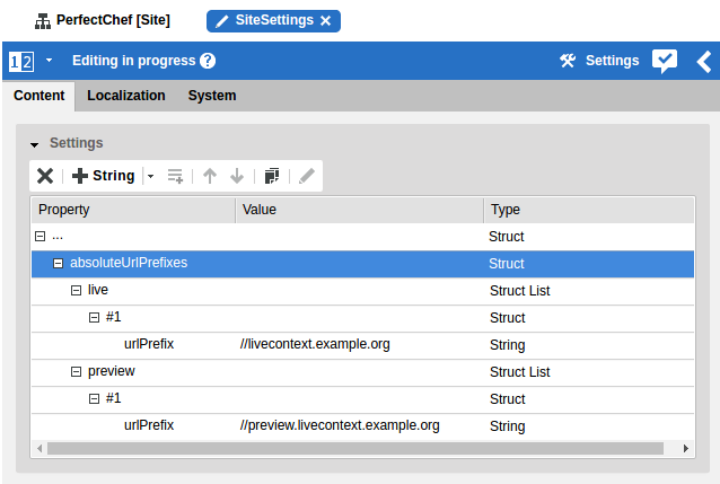


Figure 9.8. A basic absoluteUrlPrefixes Struct

The URL prefixes must be at least a scheme-relative URL (beginning with `//`).



The Blueprint features an application property `link.urlPrefixType` that determines which `absoluteUrlPrefixes` entry is effective in a particular application. You will find `link.urlPrefixType` set appropriately in the `application.properties` of all components that use the `bpbse-links-impl` module, e.g. for the `cae-live-webapp`:

```
# The live webapp builds live URLs
link.urlPrefixType=live
```

Example 9.1. Configuration of URL prefix type

While the standard Blueprint distinguishes only between preview and live URL prefixes, projects may add additional `absoluteUrlPrefixes` entries of arbitrary names for special URL prefixes and applications.

Why do we need struct lists after all, if they have only one entry? The above example is valid, but it does not show all configuration options. There are some optional features, and the equivalent complete struct would look like this:

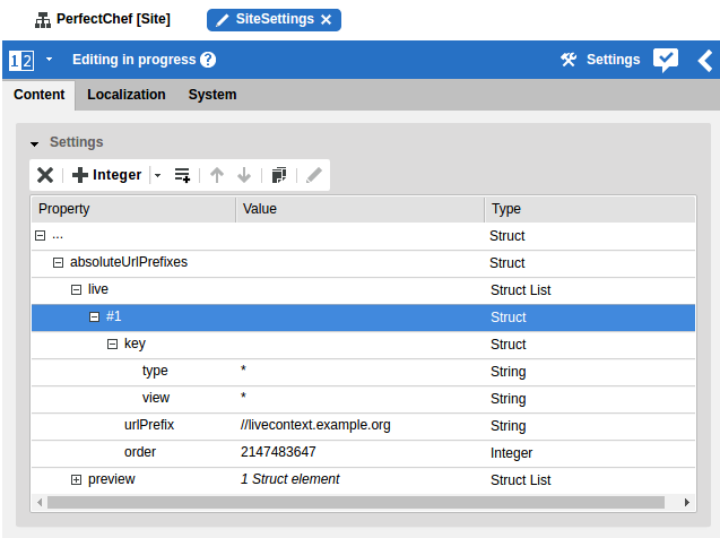


Figure 9.9. A complete `absoluteUrlPrefixes` Struct

You can declare special URL prefix rules for certain bean types or views, and you can specify an order for ambiguous rules. The default Blueprint does not make use of these options, but they reflect the format of the `key` field of the old `siteMappings` entries, so that you do not lose any features when upgrading to this mechanism.

When you start over with a fresh Blueprint and look at the `SiteSettings` documents of our example content, the configuration looks yet a little different:

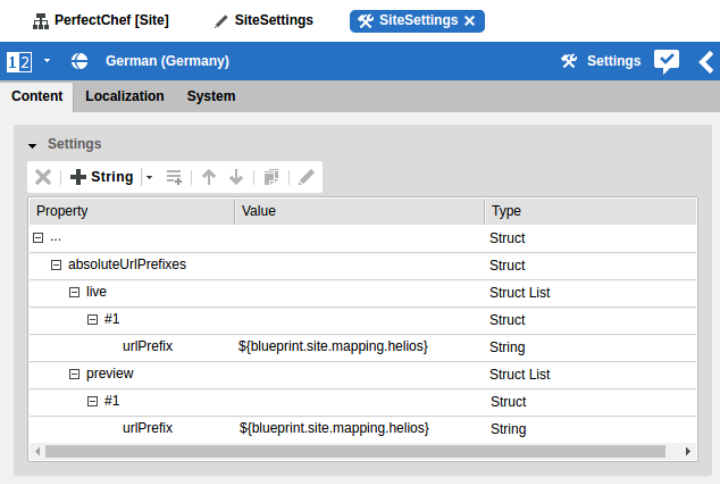


Figure 9.10. An initial `absoluteUrlPrefixes` Struct

We support a variety of deployments for various usecases like local development, production, quickrun, to name just a few. The appropriate URL prefixes in these setups vary from `//localhost:40080` to `//my.real.public.domain`. So there is no reasonable default to be hardcoded in the example content. Therefore we support application properties in the values of `urlPrefix` and use the well known `blueprint.site.mapping.*` properties which are declared in the CAEs' `application.properties` files. Initial content deployment is the only reason why these properties still exist, so you should not bother to maintain them for new sites in production repositories, but maintain the URL prefixes only in the content.

Force Scheme

In order to force a certain scheme (for example `http`, `https`, `ftp`) for a URL, two (cm) parameters must be set for the link: `absolute: true` and `scheme: <scheme-name>`.

9.8 Predefined Users

CoreMedia Blueprint provides some default users and groups that represent typical roles in an editorial staff. There are technical users with repository wide permissions and editorial users whose permissions are predominantly limited to a particular site or web presence (aside from a few exceptions like home folder access). The editorial users and groups are only available if you activate the particular extension. Depending on your specific processes and roles, the default groups may be a more or less useful starting point for a production systems. The users, however, are meant as examples only. You are supposed to replace them with users that match your actual staff. The password of all default users is the same as the name.

In the *Blueprint* workspace you will find some `test-data/users` directories (one global and some in the extensions). The XML files in those directories declare the default users, groups and rules. They can be imported with the `restoreusers` command line tool. For the initial setup of your systems, you can adapt those files to your needs. The `test-data/content` sets provide home folders with suitable editor preferences documents for the users.

The following tables show the most important default users and groups in detail.

Global

Group name	Description
staff	Root group, essential common read permissions, home folder access
administratoren	All possible permissions
developer	All possible permissions but user authorization
global-manager	Editorial permissions for global themes and settings
composer-role, approver-role, publisher-role	Publication workflow roles

Table 9.14. Global groups

While some of the global groups contain users directly, most of them serve only as parent groups for the site-specific groups.

User name	Group	Description
Adam	administratoren	Administrator: IT operations, configuration, user authorization, workflow maintenance, recovery, performance analysis

Table 9.15. Global users

User name	Group	Description
Teresa	administratoren	Online Marketing Manager: Analytics analysis, campaign management, supervision
Dave	developer	Developer: Feature development, template development, performance tuning
Amy	asset-manager	Asset Manager: managing digital assets

Since user and group names are unique within one repository, they differ for the members of the various web presences of Blueprint. The following users and groups reflect the e-Commerce web presence. The roles of the Brand web presence are basically the same, and use similar names that you will easily recognize.

e-Commerce

Group name	Description
global-site-manager	All permissions for a web presence
manager-en-US	Editorial permissions for a site, read rights for the master site
online-editor-en-US	Finegrained permissions for his particular tasks

Table 9.16. Site specific groups e-Commerce

User name	Group	Description
Rick	global-site-manager	Global site manager: organization of internal processes
Peter, Piet, Pedro, Pierre (for their respective regions)	manager-en-US, manager-de-DE, manager-es-ES, manager-fr-FR	Local content manager: management of dynamic content, targeting rules, A-B-testing, topic pages for their particular regions
George	online-editor-en-US	Online editor: writing articles, creating slideshows, editing images, tagging contents

Table 9.17. Site specific users e-Commerce

Brand web presence

Group name	Description
global-site-manager-c	All permissions for a web presence
manager-c-en-US	Editorial permissions for a site, read rights for the master site

Table 9.18. Site specific groups Brand web presence

Group name	Description
online-editor-c-en-US	Finegrained permissions for his particular tasks

User name	Group	Description
Rick C	global-site-manager-c	Global site manager: organization of internal processes
Peter C, Piet C, Pedro C, Pierre C (for their respective regions)	manager-c-en-US, manager-c-de-DE, manager-c-es-ES, manager-c-fr-FR	Local content manager: management of dynamic content, targeting rules, A-B-testing, topic pages for their particular regions
George C, Marc C	online-editor-c-en-US	Online editor: writing articles, creating slideshows, editing images, tagging contents

Table 9.19. Site specific users Brand web presence

9.9 Database Users

The following table shows the database users that are required for CoreMedia components. For MySQL and Microsoft SQL server are scripts in the workspace, that create these users.

Table 9.20. Database Users

Component	User name	Description
Content Management Server	cm7management	This database user will manage the content of the <i>Content Management Server</i> . This database will require most of the space, since content is versioned.
Master Live Server	cm7master	This database user will manage the content of the <i>Master Live Server</i> . Up to two versions of each published content will be stored.
Replication Live Server	cm7replication	This database user will manage the content of the <i>Replication Live Server</i> . Up to two version of each published content will be stored.
CAE Feeder for pre-view	cm7mcaefeeder	This database user will persist data for the <i>CAE Feeder</i> working in the management environment. Content is not versioned.
CAE Feeder for live site	cm7caefeeder	This database user will persist data for the delivery environment. Content is not versioned.

9.10 Cookies

Several customer facing modules of *CoreMedia DXP 8* use cookies to fulfill their tasks.

Blueprint delivery CAE

The Blueprint delivery CAE is configured to not write any cookies. However, session cookies CM_SESSIONID and JSESSIONID are written, when a website visitor logs into the Blueprint delivery CAE. The name of these cookies may vary, depending on the deployment scenario.

Elastic Social

Elastic Social writes only one cookie:

guid A globally unique ID to identify the user's web browser

Adaptive Personalization

In the default configuration, *CoreMedia Adaptive Personalization* writes the cookies described in the list. *CoreMedia Adaptive Personalization* can also be configured to store data in *CoreMedia Elastic Social* user profiles.

cmKeywordCookie	Scoring of keywords attached to the visited contents.
cmLastVisited	A fixed-sized list of the last visited contents.
cmLocationTaxonomiesCookie	Scoring of location taxonomies attached to the visited contents
cmReferrerCookie	Search engine and search query that lead the visitor to the site.
cmSubjectTaxonomiesCookie	Scoring of subject taxonomies attached to the visited contents.

e-Commerce

When you use e-Commerce, the IBM WebSphere Commerce Server writes cookies, documented at http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/cse_cookies.htm.

Glossary

Blob	Binary Large Object or short blob, a property type for binary objects, such as graphics.
CAE Feeder	Content applications often require search functionality not only for single content items but for content beans. The <i>CAE Feeder</i> makes content beans searchable by sending their data to the <i>Search Engine</i> , which adds it to the index.
Content Application Engine (CAE)	<p>The <i>Content Application Engine (CAE)</i> is a framework for developing content applications with <i>CoreMedia CMS</i>.</p> <p>While it focuses on web applications, the core frameworks remain usable in other environments such as standalone clients, portal containers or web service implementations.</p> <p>The CAE uses the Spring Framework for application setup and web request processing.</p>
Content Bean	A content bean defines a business oriented access layer to the content, that is managed in <i>CoreMedia CMS</i> and third-party systems. Technically, a content bean is a Java object that encapsulates access to any content, either to CoreMedia CMS content items or to any other kind of third-party systems. Various CoreMedia components like the CAE Feeder or the data view cache are built on this layer. For these components the content beans act as a facade that hides the underlying technology.
Content Delivery Environment	<p>The <i>Content Delivery Environment</i> is the environment in which the content is delivered to the end-user.</p> <p>It may contain any of the following modules:</p> <ul style="list-style-type: none">→ <i>CoreMedia Master Live Server</i>→ <i>CoreMedia Replication Live Server</i>→ <i>CoreMedia Content Application Engine</i>→ <i>CoreMedia Search Engine</i>→ <i>Elastic Social</i>

	<ul style="list-style-type: none"> → <i>CoreMedia Adaptive Personalization</i>
Content Feeder	The <i>Content Feeder</i> is a separate web application that feeds content items of the <i>CoreMedia</i> repository into the <i>CoreMedia Search Engine</i> . Editors can use the <i>Search Engine</i> to make a full text search for these fed items.
Content item	In <i>CoreMedia CMS</i> , content is stored as self-defined content items. Content items are specified by their properties or fields. Typical content properties are, for example, title, author, image and text content.
Content Management Environment	<p>The <i>Content Management Environment</i> is the environment for editors. The content is not visible to the end user. It may consist of the following modules:</p> <ul style="list-style-type: none"> → <i>CoreMedia Content Management Server</i> → <i>CoreMedia Workflow Server</i> → <i>CoreMedia Importer</i> → <i>CoreMedia Site Manager</i> → <i>CoreMedia Studio</i> → <i>CoreMedia Search Engine</i> → <i>CoreMedia Adaptive Personalization</i> → <i>CoreMedia CMS for SAP Netweaver® Portal</i> → <i>CoreMedia Preview CAE</i>
Content Management Server	Server on which the content is edited. Edited content is published to the Master Live Server.
Content Repository	<i>CoreMedia CMS</i> manages content in the Content Repository. Using the Content Server or the UAPI you can access this content. Physically, the content is stored in a relational database.
Content Server	<p><i>Content Server</i> is the umbrella term for all servers that directly access the <i>CoreMedia</i> repository:</p> <p><i>Content Servers</i> are web applications running in a servlet container.</p> <ul style="list-style-type: none"> → <i>Content Management Server</i> → <i>Master Live Server</i> → <i>Replication Live Server</i>

Content type	A content type describes the properties of a certain type of content. Such properties are for example title, text content, author, ...
Contributions	Contributions are tools or extensions that can be used to improve the work with <i>CoreMedia CMS</i> . They are written by CoreMedia developers - be it clients, partners or CoreMedia employees. CoreMedia contributions are hosted on Github at https://github.com/coremedia-contributions .
Controm Room	<i>Controm Room</i> is a <i>Studio</i> plugin, which enables users to manage projects, work with workflows, and collaborate by sharing content with other <i>Studio</i> users.
CORBA (Common Object Request Broker Architecture)	<p>The term <i>CORBA</i> refers to a language- and platform-independent distributed object standard which enables interoperation between heterogenous applications over a network. It was created and is currently controlled by the Object Management Group (OMG), a standards consortium for distributed object-oriented systems.</p> <p>CORBA programs communicate using the standard IIOP protocol.</p>
CoreMedia Studio	<p><i>CoreMedia Studio</i> is the working environment for business specialists. Its functionality covers all of the stages in a web-based editing process, from content creation and management to preview, test and publication.</p> <p>As a modern web application, <i>CoreMedia Studio</i> is based on the latest standards like Ajax and is therefore as easy to use as a normal desktop application.</p>
Dead Link	A link, whose target does not exists.
DTD	<p>A Document Type Definition is a formal context-free grammar for describing the structure of XML entities.</p> <p>The particular DTD of a given Entity can be deduced by looking at the document prolog:</p> <pre><!DOCTYPE coremedia SYSTEM "http://www.coremedia.com/dtd/coremedia.dtd"</pre> <p>There're two ways to indicate the DTD: Either by Public or by System Identifier. The System Identifier is just that: a URL to the DTD. The Public Identifier is an SGML Legacy Concept.</p>
Elastic Social	<i>CoreMedia Elastic Social</i> is a component of <i>CoreMedia CMS</i> that lets users engage with your website. It supports features like comments, rating, likings on your website. <i>Elastic Social</i> is integrated into <i>CoreMedia Studio</i> so editors can moderate user generated content from their common workplace. <i>Elastic Social</i> bases on NoSQL technology and offers nearly unlimited scalability.

EXML	EXML is an XML dialect supporting the declarative development of complex Ext JS components. EXML is Jangaroo's equivalent to Adobe Flex MXML and compiles down to Actions Script.
Folder	A folder is a resource in the CoreMedia system which can contain other resources. Conceptually, a folder corresponds to a directory in a file system.
Home Page	The main entry point for all visitors of a site. Technically it is often referred to as root document and also serves as provider of the default layout for all subpages.
IETF BCP 47	Document series of <i>Best current practice</i> (BCP) defined by the Internet Engineering Task Force (IETF). It includes the definition of IETF language tags, which are an abbreviated language code such as en for English, pt-BR for Brazilian Portuguese, or nan-Hant-TW for Min Nan Chinese as spoken in Taiwan using traditional Han characters.
Importer	Component of the CoreMedia system for importing external content of varying format.
IOR (Interoperable Object Reference)	A CORBA term, <i>Interoperable Object Reference</i> refers to the name with which a CORBA object can be referenced.
Jangaroo	<i>Jangaroo</i> is a JavaScript framework developed by CoreMedia that supports ActionScript as an input language which is compiled down to JavaScript. You will find detailed descriptions on the Jangaroo webpage http://www.jangaroo.net .
Java Management Extensions (JMX)	The Java Management Extensions is an API for managing and monitoring applications and services in a Java environment. It is a standard, developed through the Java Community Process as JSR-3. Parts of the specification are already integrated with Java 5. JMX provides a tiered architecture with the instrumentation level, the agent level and the manager level. On the instrumentation level, MBeans are used as managed resources.
JSP	JSP (Java Server Pages) is a template technology based on Java for generating dynamic HTML pages. It consists of HTML code fragments in which Java code can be embedded.
Locale	Locale is a combination of country and language. Thus, it refers to translation as well as to localization. Locales used in translation processes are typically represented as IETF BCP 47 language tags.
Master Live Server	The <i>Master Live Server</i> is the heart of the <i>Content Delivery Environment</i> . It receives the published content from the <i>Content Management Server</i> and makes it available to the CAE. If you are using the <i>CoreMedia Multi-Site Management Extension</i> you may use multiple <i>Master Live Server</i> in a CoreMedia system.

Master Site	A master site is a site other localized sites are derived from. A localized site might itself take the role of a master site for other derived sites.
MIME	With Multipurpose Internet Mail Extensions (MIME), the format of multi-part, multimedia emails and of web documents is standardised.
Personalisation	On personalised websites, individual users have the possibility of making settings and adjustments which are saved for later visits.
Projects	A project is a collection of content items in CoreMedia CMS created by a specific user. A project can be managed as a unit, published or put in a workflow, for example.
Property	<p>In relation to CoreMedia, properties have two different meanings:</p> <p>In CoreMedia, content items are described with properties (content fields). There are various types of properties, e.g. strings (such as for the author), Blobs (e.g. for images) and XML for the textual content. Which properties exist for a content items depends on the content type.</p> <p>In connection with the configuration of CoreMedia components, the system behavior of a component is determined by properties.</p>
Replication Live Server	The aim of the <i>Replication Live Server</i> is to distribute load on different servers and to improve the robustness of the <i>Content Delivery Environment</i> . The <i>Replication Live Server</i> is a complete Content Server installation. Its content is an replicated image of the content of a <i>Master Live Server</i> . The <i>Replication Live Server</i> updates its database due to change events from the <i>Master Live Server</i> . You can connect an arbitrary number of <i>Replication Live Servers</i> to the <i>Master Live Server</i> .
Resource	A folder or a content item in the CoreMedia system.
ResourceURI	A ResourceUri uniquely identifies a page which has been or will be created by the <i>Active Delivery Server</i> . The ResourceUri consists of five components: Resource ID, Template ID, Version number, Property names and a number of key/value pairs as additional parameters.
Responsive Design	Responsive design is an approach to design a website that provides an optimal viewing experience on different devices, such as PC, tablet, mobile phone.
Site	<p>A site is a cohesive collection of web pages in a single locale, sometimes referred to as localized site. In <i>CoreMedia CMS</i> a site especially consists of a site folder, a site indicator and a home page for a site.</p> <p>A typical site also has a master site it is derived from.</p>

Site Folder	All contents of a site are bundled in one dedicated folder. The most prominent document in a site folder is the site indicator, which describes details of a site.
Site Indicator	A site indicator is the central configuration object for a site. It is an instance of a special content type, most likely <code>CMSite</code> .
Site Manager	Swing component of CoreMedia for editing content items, managing users and workflows.
Site Manager Group	Members of a site manager group are typically responsible for one localized site. Responsible means that they take care of the contents of that site and that they accept translation tasks for that site.
Template	In CoreMedia, JSPs used for displaying content are known as Templates. OR In <i>Blueprint</i> a template is a predeveloped content structure for pages. Defined by typically an administrative user a content editor can use this template to quickly create a complete new page including, for example, navigation, predefined layout and even predefined content.
Translation Manager Role	Editors in the translation manager role are in charge of triggering translation workflows for sites.
User Changes web application	The <i>User Changes</i> web application is a <i>Content Repository</i> listener, which collects all content, modified by <i>Studio</i> users. This content can then be managed in the <i>Control Room</i> , as a part of projects and workflows.
Version history	A newly created content item receives the version number 1. New versions are created when the content item is checked in; these are numbered in chronological order.
Weak Links	In general <i>CoreMedia CMS</i> always guarantees link consistency. But links can be declared with the <i>weak</i> attribute, so that they are not checked during publication or withdrawal. Caution! Weak links may cause dead links in the live environment.
WebDAV	WebDAV stands for World Wide Web Distributed Authoring and Versioning Protocol. It is an extension of the Hypertext Transfer Protocol (HTTP), which offers a standardised method for the distributed work on different data via the internet. This adds the possibility to the CoreMedia system to easily access CoreMedia resources via external programs. A WebDAV enabled application like Microsoft Word is thus able to open Word documents stored in the CoreMedia system. For further information, see http://www.webdav.org .

Workflow	A workflow is the defined series of tasks within an organization to produce a final outcome. Sophisticated applications allow you to define different workflows for different types of jobs. So, for example, in a publishing setting, a document might be automatically routed from writer to editor to proofreader to production. At each stage in the workflow, one individual or group is responsible for a specific task. Once the task is complete, the workflow software ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process.
Workflow Server	The <i>CoreMedia Workflow Server</i> is part of the Content Management Environment. It comes with predefined workflows for publication and global-search-and-replace but also executes freely definable workflows.
XLIFF	XLIFF is an XML-based format, standardized by OASIS for the exchange of localizable data. An XLIFF file contains not only the text to be translated but also metadata about the text. For example, the source and target language. <i>CoreMedia Studio</i> allows you to export content items in the XLIFF format and to import the files again after translation.

Index

A

- A/B testing, 440
- access rights, 264
- actions, 294
 - webflow, 297
- Adaptive Personalization, 431
 - CAE, 433
 - CAE integration, 433
 - conditions, 437
 - content types, 253
 - context, 431
 - extension, 432
 - extension modules, 432
 - integration points, 432
 - search functions, 434
 - test user profiles, 436
- AdaptivePersonalization
 - Studio, 437
- addTranslationWorkflowPlugin, **353**
- Adobe Drive, 443
- Application
 - Architecture, 107
 - artifact, 109
 - properties, 111
- approver-role, 361
- Asset Management, 443
 - Adobe Drive connector, 458
 - AMAsset, 453
 - Asset Widget, 444
 - blob storage, 455
 - catalogPictureHandler, 448
 - content types, 453
 - download portal, 457
 - metadata, 449
- Asset Widget, 444
- assets, 248, 264

B

- b2b contract based personalization, 230
- blueprint
 - Brand Blueprint, 29
 - e-Commerce Blueprint, 29
 - removal, 44
- Brand Blueprint, 29

C

- CAE, 161
 - Maven configuration, 163
 - Performance Tests, 164
 - Personalization, 433
 - start, 89
 - use local resources, 324
- CAE Feeder, 165
 - start, 88
- catalog, 223
- Chef, 121, 128
 - Chef Development Kit, 41
 - Chef Server, 41
 - Chef Supermarket, 41
 - Encrypted Data Bags, 41
 - node.json, 129
 - solo.rb, 129
- classify, 254
- client code, 264, 291
 - merging, 293
 - minification, 293
 - performance, 292
 - preview, 292
- cm.unescape
 - FreeMarker macro, 319
- CMChannel, 266
- CMCollection, 281
- CMJavaScript, 291
- CMLocalized, 348
- CMSCSS, 291
- CMSettings, 349
- CMTaxonomy, 255
 - properties, 256
- CMTeasable, 287, 348
- CMViewtype, 282
- commerce segment personalization, 228
- common content

- types, 249
- Component
 - Artifact, 108
 - Extension, 115
 - JMX, 168
 - Logging, 166
 - start locally, 86
- composer-role, 361
- Content
 - import, 87
- content, 248
 - media, 252
- content assets
 - properties, 251
- content chooser configuration, 389
- Content Feeder
 - start, 88
- content lists, 280
- Content Server
 - start, 86
- content type
 - CMLocalized, 348
 - CMTeasable, 348-349
- content type model, **476**
 - extensions:translatable, 348-349
 - extensions:weakLink, 348
- Content Types
 - extending, 155
- content types, 264
 - assets, 264
 - client code, 264
 - navigation and page structure, 264
 - technical content types, 264
- content visibility, 301
- context, 266, 431
 - determine, 267
- CoreMedia Blueprint
 - folder structure, 264
- CoreMedia modules, 386
- country
 - locale, 332
- create content
 - add menu item, 401
- create from template
 - dialog, 406
 - new template folder, 407

- template locations, 406

CSS

- Development, 323

D

- Database
 - Configuration, 53
 - Prerequisites, 41
 - Users, 54
- database
 - users, 487
- demo data generator
 - configuration, 428
 - start, 427
- document type model (see content type model)
- Documentation, 39
- Download, 39
- dynamic templating, 288
 - add template to page, 289
 - upload templates, 289

E

- e-Commerce API, 219
- e-Commerce Blueprint, 29
- Elastic Social
 - configuration, 419
 - curated transfer, 426
 - custom information, 422
 - Demo Data Generator, 426
 - emails, 425
 - features, 418
 - mail templates, 425
- end user interactions, 294
- Environments
 - supported, 42
- extendingShopPages, 183
- Extension, 115, 147
 - activation, 117, 147
 - component, 116
 - content types, 156
 - dependencies, 151
 - Elastic Social, 149
 - Nuggad, 149
 - Personalization, 149
 - ShoutEm, 149

- Extensions, 113
- extensions:automerger, 349
- extensions:translatable, 348-349
- extensions:weakLink, 348
- external content, 395
- external library
 - configuration, 396
 - implement additional source, 397

F

- FreeMarker, 288
- FreeMarker macro
 - cm.unescape, 319

G

- global site manager, 338
- group, 361
 - approver-role, 361
 - composer-role, 361
 - publisher-role, 361

H

- Hardware Prerequisites, 40
- home page, 332-333, **333**
- Host Mappings
 - locally, 54
 - virtualized, 52

I

- IETF BCP 47, 332
- Images, 298
 - configure sizes, 298
 - default JPEG quality, 300
 - High Resolution/Retina, 300
- Installation
 - ZIP, 136

J

- Jangaroo, 113
- JavaScript
 - Development, 323
- JMeter, 164
- JMX, 168
- JNDI, 110

K

- keywords, 254

L

- language
 - locale, 332
- layout
 - localization, 279
- Library
 - catalog view, 223
- library
 - Image Gallery, 394
- License
 - configuration, 47
- link
 - weak, 336, 348, 357, 359
- link format, 478
- local site manager, **334**
- locale, **332**, 337
 - IETF BCP 47, 332
- LocaleSettings, **337**, 338
- localization, 332
- localized site, 332
- Logging, 165
 - logback, 110, 165
 - slf4j, 166

M

- mail templates, 425
- MailTemplateService, 425
- management center, 227
- master site, **333**
- Maven
 - building the workspace, 82, 113
 - changing groupId, 46
 - coremedia-application, 113
 - Extension, 115, 147
 - override-properties, 173
 - Performance Tests, 164
 - scm, 47
 - settings.xml, 45
- media content, 252
- minification, 293
- multi-site, 332

- administration, 337
- CMLocalized, 348
- CMTeasable, 348-349
- content types, **347**
- derived site, 335
- global site manager, 338
- groups, 338
- local site manager, **334**
- master site, **333**, 335
- permissions, 338
- site, **332**
- SiteModel, **342**, 363
- SitesService, **342**
- structure, 334
- translation manager role, **334**, 338, 363

N

- navigation, 265
- navigation and page structure, 264
- Nexus, 42
- Nuggad, 149

O

- Open Street Map, 440
- OpenCalais
 - disable, 261
- OpenJDK, 119
- Optimizely, 440
- Oracle JDK, 119
- OutOfMemoryException, 140

P

- Package
 - artifact, 111
- page grid, 270
 - configure new layout, 275
 - editor, 271
 - incompatible changes, 274
 - inheriting placements, 271
 - layout locations, 274
 - lock placements, 271
 - predefined layout, 273
- Performance Tests
 - CAE, 164

- Personalization
 - content types, 150
- placement, 270
- placement editor, 274
- placements
 - localization, 279
- predefined user, 484
- predefined workflows, 360
- Properties, 170
 - Default values, 171
 - Filtering, 172
 - location, 170
 - RPM deployment, 171
- Provisioning, 119
- Publication
 - bulkpublish, 91
 - publisher-role, 361

R

- rights concept, 264
- robots.txt, 303
 - example configuration, 304
- RobotsHandler, 304
- RPM, 114
 - properties, 171

S

- search, 308
- Search Engine
 - start, 86
- search functions, 434
- search landing pages, 309
 - keywords, 309
- ServerExport, **349**
- ServerImport, **349**
- Services
 - init scripts, 135
 - start, 135
 - Tomcat, 111
- settings
 - linked, 268
 - local, 268
- settings.xml, 45
- shop configuration, 207
- ShoutEm, 149

- site, **332**
 - derived site, 335
 - global site manager, 338
 - home page, 332-333, **333**, 335
 - interdependence, **336**
 - local site manager, **334**
 - locale, **332**, 337
 - LocaleSettings, **337**, 338
 - localized site, 332
 - master site, **333**, 335
 - multi-site, 332
 - site folder, 332, **333**, 335
 - site id, **333**
 - site identifier, **333**
 - site indicator, 332, **333**, 335-336, 339, 341
 - site manager group, 333, **334**, 339, 363
 - site name, **333**
 - SiteModel, 339, 341-342, **342**, 363
 - SitesService, **342**
 - translation manager role, **334**, 338, 363
 - translation workflow robot user, 339
 - Site Manager
 - start, 90
 - site manager group, 333, **334**, 339, 363
 - sitemap, 302, 306
 - maximum number of URLs, 306
 - SiteModel, **342**, 363
 - SitesService, **342**
 - Software Prerequisites, 40
 - Solr, 309
 - Studio, 157
 - bookmarks, 395
 - content chooser, 388
 - create content, 400
 - create from template, 405
 - external library, 395
 - external preview, 398
 - image link list editor, 387
 - Jangaroo, 113, 159
 - library, 394
 - Maven, 159
 - Personalization, 437
 - plugin, 157
 - plugins, 387
 - query editor, 390
 - settings, 399
 - site selection, 408
 - start, 90
 - upload files, 408
 - Studio enhancements, 387
 - suggestion strategy, 260
- ## T
- taxonomies, 254
 - as conditions for dynamic lists, 255
 - hierarchical organisation, 256
 - implement new, 260
 - implement new suggestion strategy, 260
 - location, 256
 - related content, 255
 - site specific, 261
 - taxonomy editor, 256
 - taxonomy resolver strategy, 259
 - teaser management, 286
 - technical content types, 264
 - themes
 - descriptor, 313
 - working with, 311
 - Tomcat, 114
 - development mode, 324
 - translation, 332
 - addTranslationWorkflowPlugin, **353**
 - configuration, 352
 - customization, 353, **354**
 - localization, **355**
 - Studio, 353
 - UI, **353**
 - workflow, **362**, 364
 - workflow action, 364
 - workflowForm, **353**
 - XLIFF, **351**
 - translation manager, **334**, 338, 363
 - translation manager role, **334**, 338, 363
 - translation workflow robot user, 339
- ## U
- upload files
 - configuration, 409
 - URLs, 300
 - User Changes web application, 23, 89
 - Users

- import, 87
- users
 - predefined, 484

V

- Vagrant, 120
 - destroy, 85
 - Installation, 50
 - provision, 85
 - resume, 85
 - suspend, 85
 - up, 83
- validFrom, 302
- Vanity URLs, 300
- VCS, 47
- view repositories, 290
- view type selector, 282
- view types, 282
 - localization, 280, 283
- viewType, 252
- Virtualization, 119
 - Chef, 121
 - Host Mappings, 52
 - Software, 41
 - Vagrant, 120
- visibility, 301

W

- wcs preview support, 227
- wcs workspace support, 230
- weak link, 336, 348, 357, 359
- web development workflow, 321
- web resources
 - editing, 323
 - import into repository, 326
 - local, 321
 - local resources, 323
 - preview, 326
 - release, 330
 - workspace structure, 323
- WebDAV, 442
- WebDAV Server
 - start, 90
- Webflow actions, 297
- website

- navigation, 265
- page assembly, 269
- settings, 268
- structure, 269
- website search, 308
- WebSphere
 - troubleshooting, 80
- WebSphere Commerce System
 - preview support, 227
 - workspaces support, 230
- workflow
 - action, 364
 - publication, 360
 - translation, 362, 364
- workflow action, 364
 - CompleteTranslationAction, 367
 - ExtractPerformerAction, 366-367
 - GetDerivedContentsAction, 365
 - GetSiteManagerGroupAction, 366
 - RollbackTranslationAction, 368-369
- Workflow Server
 - start, 88
- workflowForm, **353**
- workspace
 - download, 39
- Workspace
 - Build, 82
 - Configuration, 46
 - Structure, 112

X

- XLIFF, **351**, 351-352
 - emptyTransUnitMode, 352
 - ignorableWhitespaceRegex, 352
 - translation unit, 351
- XML Localization Interchange File Format, **351**